НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО

Факультет інформатики та обчислювальної техніки
(назва факультету, інституту)

Кафедра автоматизованих систем обробки інформації і управління
(назва кафедри)

"На правах рукопису"
УДК *УДК 004.021*

«До захисту допущено»
Завідувач кафедри

_____ О.А.Павлов
(підпис) (ініціали, прізвище)

“____” _____ 20*19* р.

МАГІСТЕРСЬКА ДИСЕРТАЦІЯ

на здобуття ступеня магістра

за спеціальністю *126 Інформаційні системи та технології*
(код та назва спеціальності)

ОПП *Інформаційні управляючі системи та технології*
(код та назва спеціалізації)

на тему: *Ефективні алгоритми обчислювання криптопримітивів в паралельній системі обчислень*

Виконав: студент *VI* курсу групи ІС-82мп
(шифр групи)

Рогоза Антон Віталійович _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник *проф., д.ф.-м.н., проф. Задірака В.К.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант *к.т.н., доц. Жданова О.Г.* _____
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент *доц каф.ТК, к.т.н., доц. Лісовиченко О.І.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

РЕФЕРАТ

Магістерська дисертація: 94 с., 30 рис., 28 табл., 1 додаток, 24 джерел.

Актуальність. Громіздке впровадження IT робить актуальною проблему захисту інформації. Статистика показують, що лише деякі власники своїх фірм в цій галузі вважають свою компанію такою, яка готова протистояти сучасним інформаційним загрозам.

У цьому дослідженні вирішено питання підвищення ефективності алгоритмів шифрування та дешифрування інформації. Підвищення кількості цифрових документів та комерційних операцій створюють гостру потребу в наявності алгоритмів шифрування. Безпечні криптографічні алгоритми є дуже трудомісткими і їх ефективна реалізація необхідна для програм. У цьому дослідженні представлена паралельна реалізація та аналіз різних алгоритмів шифрування, таких як: AES (Advanced Encryption Standard), Blowfish, Twofish, DES (Data Encryption Standard), Serpent, Triple DES та RSA. Паралельна реалізація алгоритмів створена для підвищення ефективності алгоритмів. Експерименти показують, що послідовна реалізація значно поступається паралельній в часу виконанні. Алгоритми порівнюються на основі часу шифрування та дешифрування інформації та їх прискорення. Результати показують, що Triple DES є більш ефективним алгоритмом серед досліджуваних алгоритмів.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалась на філії кафедри автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського» в рамках теми «Розробити оптимальні за точністю та швидкодією криптографічні алгоритми, розпаралелити їх та зробити їх порівняльний аналіз».

Мета: збільшити прискорення алгоритмів шифрування та дешифрування інформації□.

Для досягнення задоної мети необхідно виконати такі завдання:

– зробити огляд існуючих рішень;

– розробити алгоритми на основі існуючих засобів шифрування та дешифрування інформації, розпаралелити кожен із запропонованих алгоритмів;

– розробити програмну реалізацію поданих алгоритмів;

– здійснити порівняльний аналіз послідовного та паралельного виконання різних алгоритмів.

Об'єкт дослідження – шифрування інформації та її дешифрування.

Предмет дослідження – алгоритми за домогою яких можна здійснювати шифрування та дешифрування інформації.

Наукова новизна одержаних результатів полягає у розпаралелюванні існуючих криптографічних алгоритмів та здійсненні їх порівняльного аналізу.

Публікації. Матеріали роботи опубліковані в Міжнародній науковій конференції "iScience", в Міжнародній конференції "UKRLOGOS", у Всеукраїнській науково-практичній конференції молодих вчених та студентів "Інформаційні системи та технології управління" та в Міжнародній науковій інтернет-конференції "Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення".

ЕФЕКТИВНІ МЕТОДИ ОБЧИСЛЮВАННЯ КРИПТОПРИМІТИВІВ В ПАРАЛЕЛЬНІЙ СИСТЕМІ ОБЧИСЛЕНЬ.

ABSTRACT

Master's thesis: 94 pp., 30 fig., 22 tab., 1 app., 24 sources.

The relevance. The cumbersome implementation of IT makes it an urgent problem to protect information. Statistics show that only some business owners in this industry consider their company to be able to withstand today's information threats.

This study addresses the issue of improving the efficiency of encryption and decryption algorithms. Increasing numbers of digital documents and commercial transactions create an urgent need for encryption algorithms. Secure cryptographic algorithms are very time consuming and their effective implementation is necessary for programs. This study presents the parallel implementation and analysis of various encryption algorithms such as: AES (Advanced Encryption Standard), Blowfish, Twofish, DES (Data Encryption Standard), Serpent, Triple DES and RSA. Parallel implementation of algorithms is created to increase the efficiency of algorithms. Experiments show that a consistent implementation is significantly inferior to a parallel one at run time. The algorithms are compared based on the time of encryption and decryption of information and their acceleration. The results show that Triple DES is a more efficient algorithm among the studied algorithms.

Relationship with working with scientific programs, plans, topics. The work was performed at the branches of the Department of Automated Information Processing and Control Systems of the National Technical University of Ukraine «Kyiv Polytechnic Institute. Igor Sikorsky» within the theme «Develop optimal cryptographic algorithms for accuracy and speed, parallelize them and make their comparative analysis».

Objective: To increase the acceleration of information encryption and decryption algorithms.

To achieve the desired goal, you must perform the following tasks:

– review existing solutions;

– to develop algorithms on the basis of existing means of encryption and decryption of information, to parallelize each of the proposed algorithms;

– to develop software implementation of the given algorithms;

– to perform comparative analysis of sequential and parallel execution of different algorithms.

The object of the study is to encrypt information and decrypt it.

The subject of the study - algorithms by which you can perform encryption and decryption of information.

The scientific novelty of the obtained results is the parallelization of existing cryptographic algorithms and their comparative analysis.

Publications. The materials have been published in the International Scientific Conference “iScience”, in the International Conference “UKRLOGOS”, in the All-Ukrainian Scientific and Practical Conference of Young Scientists and Students “Information Systems and Technologies of Management” and in the International Scientific Internet Conference “Information Society: Technological, Economic and technical aspects of becoming".

EFFECTIVE ALGORITHMS FOR CALCULATING CRYPTO PRIMITIVES IN A PARALLEL COMPUTING SYSTEM.

ЗМІСТ

ВСТУП

Деякі секрети потрібно зберігати ще з колиски цивілізації і з винахідом писемності, людство намагалося приховати інформацію від сторонніх очей. Історичні дані свідчать про те, що криптографічні схеми
були використані багатьма ранніми цивілізаціями, такими як: єгиптяни, євреї та греки. У середньовічну епоху були виявлені досить складні шифри,
але до Першої та Другої світових воєн, вони розвивались розвивались спеціальних підходів.

Криптологія - це математична наука про шифри, і, до речі, її етимологією є грецька kriptós logia, що означає «вивчення таємниць». Криптологія - це термін для криптографії та криптоаналізу двох об’єктів.

Ці об’єкти пов'язані між собою так само, як гра кота з мишею; поки криптографія - це мистецтво побудови алгоритмів та протоколів; криптоаналіз приймає суперечливу точку зору криптології при руйнуванні чи аналізі
безпеки криптографічних конструкцій.

До світанку цифрової революції в основному використовувалась криптографія,
яка забезпечує секретності при передачі секретної інформації між двома сторонами через незахищений канал, такий як радіозв'язок або а телефонна лінія. Але в сучасну епоху комп'ютерів та її величезних потоків інформації криптографія швидко розвивалася і стала чимось дивовижним.

З одного боку, це стало необхідністю безпеки даних, автентифікації, цифрових підписів тощо - щось електронна комерція, банківські та кредитні картки не могло існувати без. З іншого боку, криптографія стала інструментом в політиці. Анонімізація сильною криптографією дозволяє журналістам зберегти цілісність. Це також породило рухи наприклад, криптоанархізм та криптокапіталізм, які використовують криптографію для захищення конфіденційність та політичної свободи. Одним із стовпів криптографії є, безперечно, принцип Керкхоффа, прийнятий
більшістю криптографів. Принцип Керкхоффа по суті стверджує, що безпека криптографічної конструкції повинна залежати лише від секретного ключа.

Будь-яку частину конструкції слід вважати відомою потенційному противнику.

Грубо кажучи, мета криптографічної схеми шифрування полягає в тому, щоб зробити повідомлення якимось не зрозумілим таким чином, щоб отриманий шифротекст виявлявся для супротивника малозрозумілим або взагалі не містив інформації про вміст переданого тексту, дозволяючи одержувачу перетворити текст за допомогою спеціального алгоритму для отримання оригіналу повідомлення. Ці дві процедури називаються шифруванням та дешифруванням.

Схеми шифрування (і криптографії в цілому) можна розділити на симетричні та асиметричні (або відкритий ключ). Криптографію не слід плутати зі стеганографією, що є терміном для приховування інформації. Симетричне шифрування дозволяє двом сторонам безпечно спілкуватися, але їм потрібно заздалегідь узгодити загальний секретний ключ. Той самий ключ використовується для шифрування та дешифрування.

В криптографичній термінології вихідне повідомлення називають відкритим текстом (plaintext або cleartext). Зміна вихідного тексту так, щоб скрити від інших його склад, називають шифруванням (encryption). Зашифроване повідомлення називають шифротекстом (ciphertext). Процес, при якому із шифротексту видаляється відкритий текст називають дешифруванням (decryption). Переажно в процесі шифрування і дешифрування використовується деякий ключ (key) і алгоритм забезпечує, що дешифрування можна зробити лише знаючи ключ. Криптографія – це наука про те, як забезпечити секретність повідомлення.

Криптоаналіз – це наука про те, як відкрити шифроване повідомлення, так як відкрити текст не знаючи ключ. Криптографією займаються криптографи, а криптоаналізом займаються криптоаналітики. Основна класифікація криптографічних систем: Перш ніж приступитися до класифікації криптологічних систем слід зробити деякі зауваження. По етапах розвитку можна виділити три періоди розвитку:

– перший етап – етап донаукової криптології (до 1949 р.);

– другий етап – етап наукової криптології із секретними ключами (з 1949 р. по сімдесяті роки);

– третій етап – етап наукової криптології з використанням ЕОМ (із сімдесятих по теперішній час). Звичайно, такий розподіл достатньо умовний, але воно враховує основні події, що вплинули на розвиток криптології як науки. Якщо в стародавньому світі криптологією займалися як мистецтвом, то зараз криптологія стала наукою. Прорив у криптоаналізі був зроблений з моменту виникнення ЕОМ. Слід зазначити, що перші досліди такого роду були початі під час другої світової війни. В Англії в 1942 році вступили в стрій кілька спеціалізованих для злому шифрів ЕОМ, спеціально створених для цього Аланом Тюрінгом. Це була перша у світі досить швидкодіюча ЕОМ за назвою “Колос”. За допомогою цієї машини англійські криптоаналітики менше ніж за день могли розколоти будь-яку шифровку німецької шифрувальної машини Энігма. Створення персональних ЕОМ відкрило нову сторінку в криптології. З однієї сторони різко зросли можливості криптоаналітиків. Це в першу чергу стосується криптоаналізу за допомогою простого перебору. З іншої сторони з’явились практично необмежені можливості у шифрувальників, які, використовуючи можливості ПЕОМ, створювати шифри, що практично не розкриваються.

1 ОГЛЯД СУЧАСНОЇ КРИПТОГРАФІЇ ТА ЇЇ ПРОБЛЕМАТИКИ

1.1 Симетричні та асиметричні криптоалгоритми

Симетричне шифрування – це найпростіший вид шифрування, який включає лише один секретний ключ для шифрування та дешифрування інформації. Симетричне шифрування - стара і найвідоміша методика. Він використовує секретний ключ, який може бути або числом, словом, або рядком випадкових літер. Це змішаний з простим текстом повідомлення, щоб змінити вміст певним чином. Відправник та одержувач повинні знати секретний ключ, який використовується для шифрування та дешифрування всіх повідомлень. Blowfish, AES, RC4, DES, RC5 та RC6 - приклади симетричного шифрування. Найпоширенішим симетричним алгоритмом є AES-128, AES-192 та AES-256 [1].

Основним недоліком симетричного шифрування ключа є те, що всі залучені сторони повинні обміняти ключ, який використовується для шифрування даних, перш ніж вони зможуть їх розшифрувати.

Асиметричне шифрування також відоме як криптографія відкритого ключа, що є порівняно новим методом порівняно з симетричним шифруванням. Асиметричне шифрування використовує дві клавіші для шифрування простого тексту. Секретні ключі обмінюються через Інтернет або велику мережу, щоб зловмисники не використовували ключі. Важливо зазначити, що кожен із секретним ключем може розшифрувати повідомлення, і саме тому асиметричне шифрування використовує два пов'язані ключі для підвищення безпеки. Публічний ключ надається у вільний доступ усім, хто захоче надіслати вам повідомлення. Другий приватний ключ зберігається в таємниці, щоб про нього могли знати тільки ви. Класифікацію алгоритмів шифрування зображено на рисунку 1.1.

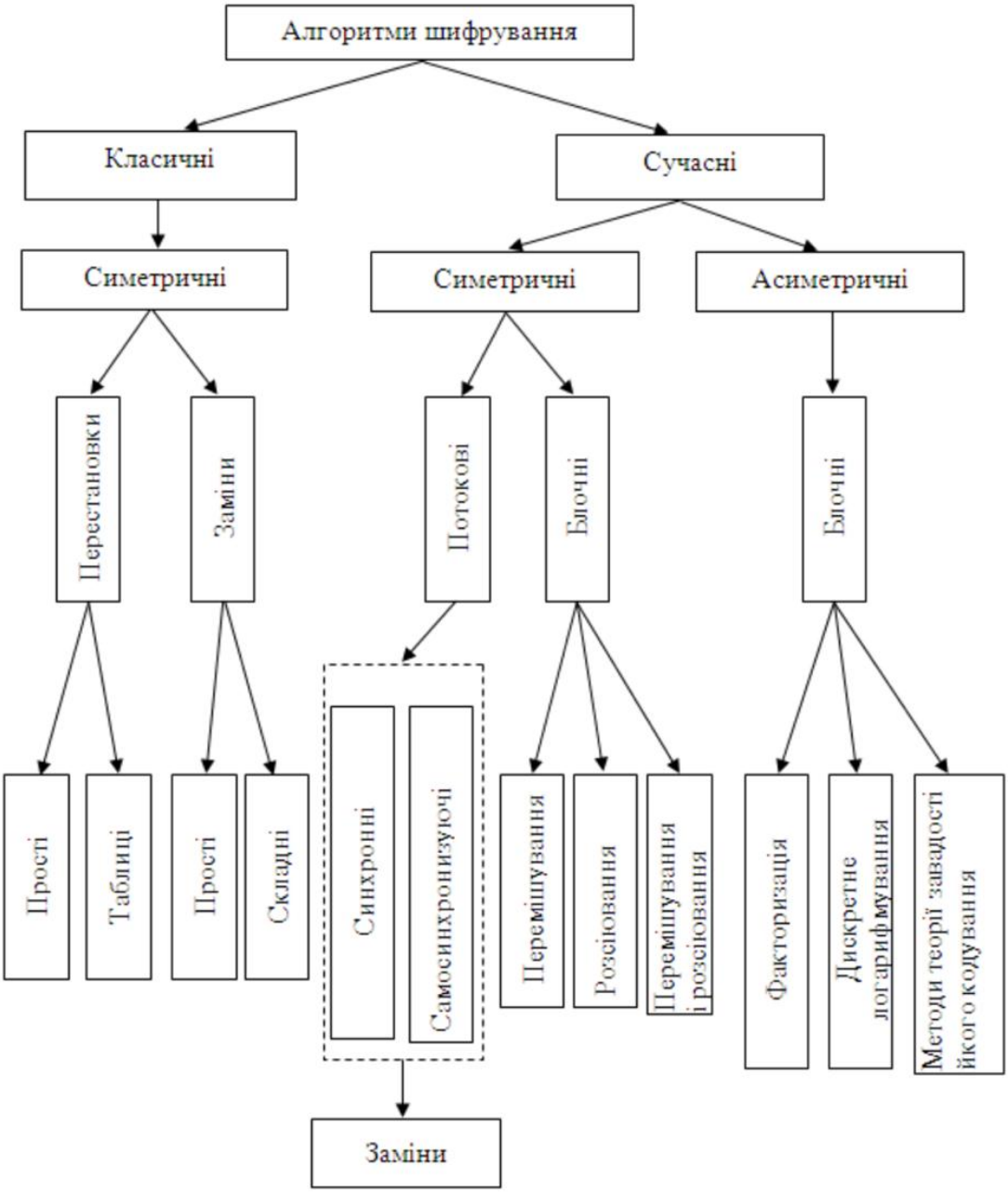


Рисунок 1.1 – Класифікація алгоритмів шифрування

Повідомлення, яке шифрується за допомогою відкритого ключа, можна розшифрувати лише за допомогою приватного ключа, тоді як повідомлення, зашифровані за допомогою приватного ключа, можна розшифрувати за допомогою відкритого ключа. Безпека відкритого ключа не потрібна, оскільки є загальнодоступною та може передаватися через Інтернет.

Асиметричне шифрування використовується в основному в повсякденних каналах зв'язку, особливо через Інтернет. Популярний алгоритм шифрування асиметричного ключа включає RSA, DSA, методи еліптичної кривої [2].

Різниця між симетричним та асиметричним шифруванням така. Симетричне шифрування використовує єдиний ключ, який потрібно ділити між

людьми, яким потрібно отримувати повідомлення, тоді як асиметричне шифрування використовує пару відкритого ключа та приватний ключ для шифрування та дешифрування повідомлень під час спілкування. Симетричне шифрування - це стара методика, тоді як асиметричне шифрування є відносно новою. Асиметричне шифрування було введено для доповнення притаманної проблеми необхідності спільного використання ключа в симетричній моделі шифрування, усуваючи необхідність ділитися ключем за допомогою пари публічно-приватних ключів. Асиметричне шифрування займає порівняно більше часу, ніж симетричне шифрування.

Вираз "комп'ютерна безпека" використовується для опису процесу збереження конфіденційності інформації, що зберігається на комп'ютерах. Однак, в даний час з масовим розповсюдженням комп'ютерів по всьому світу та з розширеним доступом до Інтернету, комп'ютерна безпека зараз займається іншими проблемами, такими як захист конфіденційності користувачів та інтелектуальної власності, які стали серйозними проблемами через збільшення кількості людей, які використовують комерційні технології та послуги цифрового банкінгу, а також через найбільш організовані злочини в Інтернеті.

Це призведе до інформаційної безпеки з точки зору комп'ютерів, тому принципи захисту інформації повинні бути задовільними в кожній організації чи навіть у кожній комп'ютерній системі. Таким чином, інформаційна безпека несе відповідальність за захист даних від несанкціонованого доступу. Інформаційна безпека складається з трьох основних опор: конфіденційність, цілісність та доступність. Конфіденційність означає, що стороннім сторонам забороняється доступ до інформації, тоді як цілісність полягає в тому, щоб дані не були змінені після їх надсилання та отримання, тобто не були змінені третьою стороною. Доступність гарантує, що дані будуть доступні у будь-який час, якщо запитуюча сторона має право.

Конфіденційність та цілісність реалізуються за допомогою шифрування, однак досягнення доступності має здійснюватися іншою функцією. Існує багато алгоритмів, які можна використовувати для шифрування та дешифрування даних. У минулому сила шифрування базувалася на секреті алгоритму; Однак

багато сценаріїв показали, що це неправда, оскільки використання одного і того ж алгоритму весь час, незалежно від його потужності, знизить безпеку системи.

Алгоритми шифрування можна класифікувати на дві основні категорії: алгоритми симетричних ключів та алгоритми відкритого ключа. Ключова особливість алгоритмів симетричних ключів полягає в тому, що ключ дешифрування можна обчислити з ключа шифрування, однак вони є рівними у більшості практичних застосувань. Симетричні ключові алгоритми можна розділити на дві гілки: блок-шифрова схема та схема шифрування потоку. Схема шифрування блочного шифру розбиває повідомлення на блоки із певною довжиною та шифрує кожен окремо. У своєму найпростішому вигляді, коли розмір кластера дорівнює одиниці, схема блокового шифру стає схемою потокового шифру. Схема шифрування потоку має перевагу перед схемою криптографії кластера. У схильній до помилок передачі передавач може передавати неправильні дані, не впливаючи на процес дешифрування, і його можна легко використовувати при телефонних розмовах та бездротових мережевих з'єднаннях. Крім того, він вважається менш безпечним, ніж схема шифрування блоків. Navalgund в 2013 році запропонував версію паралелізації алгоритму AES, використовуючи (OpenMP) модель програмування для вираження складних конвеєрних обчислень. Пропонована версія паралелізації алгоритму AES вдосконалена на рівні даних та на рівні управління. Відповідно до паралельних моделей обчислення, незалежні частини алгоритмів повинні бути визначені, а потім призначені для роботи в окремих потоках. Алгоритми поділяються на паралелізуючі та непаралелізуючі частини. Паралельна частина та непаралельна частина поєднуються за допомогою моделі вилки з'єднання. Ця робота зосереджена на паралелізації на рівні даних та на рівні контролю [3].

Nagendra запропонував паралелізацію алгоритму AES у своїй роботі, покращивши продуктивність AES з використанням паралельних обчислень, що залежить від підходу розділення та володарювання. Такий підхід застосовується при паралельних обчисленнях для паралельного вирішення алгоритмів шляхом розподілу на кілька підзадач на доступні одиниці обробки. Текстовий файл подається як вхід, потім він розкладається на певну кількість

блоків. Кожен блок виконується в одному ядрі. Реалізація алгоритму криптографії здійснюється на двоядерному процесорі. Система спеціально призначена для двоядерних процесорів, але останні процесори мають багато ядер, завдяки чому продуктивність системи не сильно покращується. Система спеціально розроблена для алгоритму шифрування в симетричних криптосистемах. Але більшість додатків в Інтернеті використовують лише асиметричну криптографію для досягнення конфіденційності, автентичності та цілісності. Блоки текстового файлу виконуються в двоядерному процесорі, так що ресурси CPU не використовуються повністю в циклічних операторах. Текстовий файл виконується паралельно, але вихідний код не паралельний.

Паралельна обробка – це форма обчислень, в якій багато обчислень проводяться одночасно, діючи за принципом, що великі проблеми часто можна розділити на менші, які потім вирішуються одночасно («паралельно»). Існує кілька різних форм паралельних обчислень: бітовий рівень, рівень інструкцій, паралелізм даних та завдань. Оскільки споживання електроенергії (а отже, і вироблення тепла) комп'ютерами стало проблемою, паралельні обчислення стали домінуючою парадигмою в архітектурі комп'ютерів, головним чином у вигляді багатоядерних процесорів. Паралельні комп’ютери можна приблизно класифікувати за рівнем, на якому є апаратне забезпечення, яке підтримує паралелізм, при цьому багатоядерні та багатопроцесорні комп'ютери мають декілька елементів обробки в одній машині, тоді як кластери, MPP (Multi Parallel Processing) та сітки використовують декілька комп'ютерів для роботи над одним завданням. Спеціалізовані паралельні комп'ютерні архітектури іноді використовуються поряд із традиційними процесорами для прискорення конкретних завдань.

Основні частини дослідження, які можна розділити на дві істотні частини: безпеку та паралельні частини. Основна увага приділялася опису криптографічним алгоритмам, що спираються на математику, і потрібні математичні знання з конкретною кінцевою метою, щоб мати можливість їх програмувати. Особливо це актуально, коли цільовою місією є паралелізація цих алгоритмів, які потребують дуже глибокого розуміння не лише їх кроків та

етапів, але й того, як вони працюють. Пропонована паралельна система фокусується на технологіях, доступних для побудови паралельної програми. Крім того, він зосереджений на ефективності, яка може бути застосована для вдосконалення паралельного алгоритму [4].

1.2 Опис визначень та термінів

Відправляючий текст – оригінальне повідомлення у формі, зрозумілій всім сторонам, включаючи відправника повідомлення та одержувача, шифротекст також зрозумілий тому, хто підслуховує спілкування між цими двома об’єктами. З іншого боку, шифртекст - це зашифроване повідомлення, яке не повинні розуміти інші сторони, окрім відправника та одержувача повідомлення. Простий текст перетворюється в шифротекст процесом шифрування, а шифротекст перетворюється назад у відкритий текст шляхом дешифрування.

Система виконання шифрування та дешифрування повідомлень називається шифром або криптографічною системою. Наука та мистецтво побудови криптографічних систем називається криптографією. Зловмисник намагається з'ясувати оригінальне повідомлення в прямому тексті (або ключ), не знаючи всіх деталей системи. Цей напрямок наукових досліджень називається криптоаналізом, а область дослідження, що поєднує криптографію та криптоаналіз, називається криптологією. Існує кілька різних криптографічних примітивів, таких як потокові шифри та блокові шифри. Ключ у криптографії - це сутність, яка використовується для налаштування використовуваного шифру, щоб лише ті сторони, які мають правильний ключ, могли викрити оригінальне повідомлення з простого тексту із зашифрованого. Криптографічна система, яка використовує один і той же ключ як зі сторони передавача, так і зі сторони приймача, називається шифром загального доступу, секретним ключем або симетричним ключем. Система, яка використовує різні ключі для шифрування та дешифрування, називається відкритим ключем або асиметричним ключем.

Паралельне програмування - це форма обчислень, яка спирається на одночасне виконання компонентів програми, це підвищує продуктивність

програми та скорочує час виконання. По-перше, розробка процесорів, прийняла закон Мура для підвищення продуктивності за рахунок збільшення кількості транзисторів в інтегрованій мікросхемі, а потім була прийнята інша методика, яка намагалася реалізувати неявний паралелізм, виконавши більше однієї інструкції за один цикл, така як зростаюча частота процесорів. Однак ці технології також досягли своїх меж, коли почали виникати проблеми, пов'язані зі споживанням енергії та тепла.

В даний час для досягнення мети застосовуються багатоядерні технології при збільшенні кількості ядер в одному процесорі. Однак, щоб максимально використати ці процесори, додаток має бути спроектовано синхронно; це потреба в паралельних технологіях та на високому рівні паралельних мов програмування, щоб полегшити завдання програмісту розробити багатопотокові або багатопроцесорні програми.

Багатоядерна система - це система, що складається з двох або більше ядер у межах одного процесора. Тут ядро - це не що інше, як блок обробки або виконання. Багатоядерна архітектура складається з двох або більше ядер обробки на одній мікросхемі. Отже, його також називають чіпом. У дизайні багатоядерної архітектури кожне ядро має власний конвеєр виконання, і кожне ядро має ресурси, необхідні для запуску, не блокуючи ресурси, необхідні для інших програмних потоків. На рисунку 1.2 показана архітектура багатоядерних систем, в якій є n кількість процесорних ядер, інтегрованих в єдиний чіп.

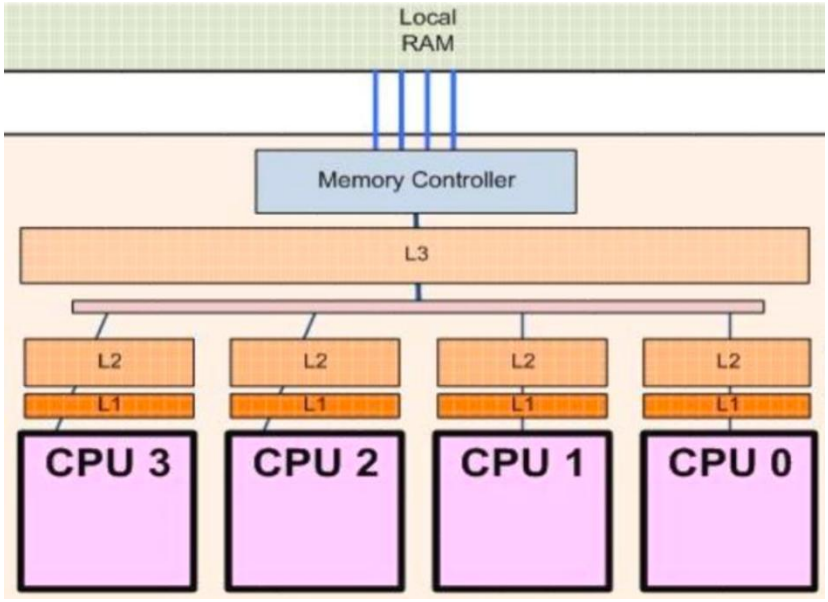


Рисунок 1.2 – Архітектура багатопроцесорних систем

Кожні ядра обробки мають власний приватний кеш L1 і поділяють загальний кеш L2. Пропускна здатність між кешем L2 та основною пам'яттю поділяється на всі ядра обробки.

Двоядерний процесор містить два ядра, чотирьохядерний процесор містить чотири ядра тощо. Багатоядерний процесор підтримує багатопроцессорність в єдиний фізичний пакет. Різні ядра в багатоядерній системі можуть бути з'єднані між собою вільно або щільно. Багатоядерна система підтримує концепцію одночасної багатопотоковості. Це дозволяє одночасно виконувати кілька незалежних потоків на одному ядрі. Так, у багатоядерних системах без потоків, є можливість виконувати одночасно кілька задач.

1.3 Принципи шифрування

У шифрованому симетричному ключі той самий ключ використовується як для шифрування, так і для дешифрування. Функція шифрування для шифру симетричного ключа - бієктивна функція тобто $X \rightarrow Y$, коли кожному елементу y з множини Y зіставлений один і лише один елемент x з множини X, і f(x) = y, що перетворює вхідне повідомлення простого тексту в шифротекст. Повідомлення в простому тексті належить до кінцевого простору повідомлень M, а повідомлення шифртексту до кінцевого простору повідомлень C [1].

Ключова функція шифрування може бути представлена у вигляді

$$c = Ee\ (m);\ m \in M,\ e \in Ke\ , \qquad (1.1)$$

де e - ключ шифрування, m - вхідне повідомлення в простому тексті, а c - вихідне повідомлення з шифротекстом. Ключ e вибирається з доступного простору Ke і визначає, як функція Ek відображає повідомлення в простому тексті на повідомлення шифротексту. Хороший шифр повинен виконувати різну трансформацію з кожним різним ключем від клавішного простору.

Аналогічно, функція дешифрування є бієктивною функцією для перетворення вхідного шифротексту в повідомлення в простому тексті. Функція дешифрування може бути представлена як

$$m = Dd\ (c);\ c \in C,\ d \in K\ , \qquad (1.2)$$

де d в ключі дешифрування, c - вхідне повідомлення про шифротекст, а m - вихідне повідомлення в простому тексті. Функція дешифрування з правильним відповідним ключем дешифрування d є зворотньою функцією шифрування з відповідним ключем шифрування

$$\text{e}, m = Dd\ (c) = E\text{ - }1\ (c).\ Kd\ Ke\ , \qquad (1.3)$$

поєднання цих відповідних ключів із функціями шифрування та дешифрування визначає криптографічну систему чи шифр. У шифрі симетричного ключа ключ шифрування може легко перетворитись у ключ дешифрування і навпаки. Як правило, ключ шифрування та ключі дешифрування однакові, e = d.

Для шифрів асиметричного ключа ключ дешифрування відрізняється від ключа шифрування, і ключ шифрування неможливо легко перетворити в ключ дешифрування без додаткової інформації. Шифри з асиметричним ключем дозволяють оприлюднити ключ шифрування, тому кожен, хто має доступ до відкритого ключа, може видавати повідомлення про шифротекст, що лише власник «приватного» ключа дешифрування може перетворити у простий текст. Шифрування симетричного ключа, навпаки, вимагає, щоб ключ передався певним способом між відправником і одержувачем, не роблячи ключ відкритим. У цьому світлі шифрування асиметричного ключа може здатися більш бажаним для зв'язку, оскільки ключі шифрування можуть бути оприлюднені, не ризикуючи витоком критичної інформації. Однак шифри з асиметричним ключем потребують значної обчислювальної обробки, і в результаті вони принаймні на 2–3 порядки повільніші, ніж симетричні ключові шифри. Тому шифри симетричного ключа використовуються для масового шифрування великих даних. Шифри асиметричного ключа та пов'язані з ними схеми обміну ключами використовуються для передачі спільного симетричного ключа між сторонами, що його надсилають і приймають.

Сучасні шифри розроблені так, що безпека схеми шифрування залежить лише від секретності ключа шифрування. Алгоритм шифрування повинен бути сконструйований таким чином, щоб алгоритм міг потрапити до рук противника, не ризикуючи безпекою. Це один із принципів дизайну шифрів, який був представлений Керкхоффом (Керкхофф, 1883), і тепер він відомий як принцип

Керкхоффа. Причина, чому такий принцип є цінним, полягає в тому, що алгоритми важко змінити після розгортання для використання в програмному та апаратному забезпеченні [5].

Якщо безпека залежить від ключа, можна використовувати один і той же алгоритм набагато довший час і зробити шифрування більш практичним. Дозволяючи алгоритму шифрування бути загальнодоступним, алгоритм також привертає увагу при дослідженні криптоаналізу. Таким чином алгоритм отримує громадський контроль і можливі недоліки шифру можуть бути виявлені ще до того, як шифр буде введений у використання.

Існують різні класи шифрування симетричних ключів, такі як блокові шифри та потокові шифри. Блокові шифри шифрують повідомлення з відкритим текстом фіксованого розміру для повідомлень шифротексту однакового розміру. Потокові шифри з іншого боку шифрують повідомлення по одному біту або байту за один раз. Зазвичай шифрові потоки генерують псевдовипадковий потік ключів на основі використовуваного ключа шифрування, який потім піддається операції XOR з повідомленням у простому тексті для отримання повідомлення про шифротекст [6].

1.4 Блокові шифри

Блокові шифри можна вважати основним складовим блоком схем шифрування симетричного ключа. Оскільки продуктивність шифрування є критичною, особливо на серверній частині зв'язку, хороші блокові шифри розроблені таким чином, щоб вони були швидкими в програмному та апаратному забезпеченні. Вхід для шифрування блок-шифру - це повідомлення фіксованого розміру, яке перетворюється на повідомлення шифротексту. Ця фіксована довжина повідомлення називається розміром блоку. Найпоширеніші розміри блоків, які використовуються в поточних блокових шифрах, - це 64 біти та 128 біт [7].

Блок-система шифрування містить функцію шифрування, функцію дешифрування та ключ шифрування. Довжина ключа може змінюватися, і прийнята довжина ключа залежить від алгоритму шифрування. Для кожної різної клавіші функція шифрування блочного шифру буде виконувати різну

перестановку на одному вхідному блоці простого тексту. Якщо ключ шифрування є секретним для сторонніх, перестановка видається випадковою. Тому блок-шифри можна описати як псевдовипадкову перестановку. Причина, чому блокові шифри є псевдовипадковими, а не справді випадковими, полягає в тому, що, знаючи ключ, шифротекст вже не є випадковим. Хоча побудова функції шифрування відрізняється для кожного алгоритму шифрування, існують основоположні принципи, на яких базується проектування та побудова блокових шифрів. Фундаментальні криптографічні прийоми плутанини та дифузії були введені Шенноном (Shannon, 1949). Метою плутанини в шифрованій конструкції є ускладнення статистичного аналізу шифротексту для криптоаналітиків. Сильна плутанина в шифрі може бути здійснена за допомогою складного нелінійного алгоритму заміщення. Метою дифузії є поширення інформації блоку простого тексту на всю ширину блоку шифротексту. У двійковому шифрі це означає, що зміна одного біту простого тексту призводить до зміни більшості біт шифротексту. Таке поширення інформації при простому тексті по всій ширині шифротексту значно ускладнює криптоаналіз.

Сильні блокові шифри, як правило, будуються з більш слабких частин, раундами. Структура такого сильного блочного шифру така, що слабкий блоковий шифр – раунд – повторюється кілька разів різними підключами або круглими клавішами. Ці кілька раундових ключів виводяться з ключа шифрування блочного шифру, який іноді називають головним ключем. Процес побудови під-ключів з головного ключа називається розкладом ключів. Міцність блок-шифру зростає зі збільшенням кількості раундів із збільшенням кількості дифузії та плутанини.

1.5 Висновки до розділу

У даному розділі було подано огляд сучасної криптографії загалом, виділено вузькі місця з якими потрібно боротися та запропоновано можливі рішення вирішення поширених проблем. Проведено аналіз шифрування та шляхи його прискорення.

2 РЕАЛІЗАЦІЯ БЛОК-ШИФРІВ

У цьому розділі ми наводимо короткі описи п'яти блокових шифрів, реалізованих у цій тезі. Кожна з цих шифрів має власну структуру; ця варіація робить більш детальну експертизу цікавою з точки зору швидкої реалізації. Кожна з цих блокових шифрів використовується на практиці хоча б певною мірою. AES, DES та Triple DES схвалили різні організації зі стандартизації.
AES широко використовується в різних вбудованих додатках, таких як зашифровані Wi-Fi з'єднання. Blowfish, Twofish та Serpent також використовуються у кількох різних криптографічних продуктах [8].

2.1 Стандарт попереднього шифрування (AES)

У 2000 році шифр Rijndeal був оголошений переможцем конкурсу Advanced Encryption Standard. Конкурс проводився Національним інститутом стандартів і технологій для пошуку нових стандартних блокових шифрів для уряду США. П'ятнадцять пропозицій було подано криптографічною спільнотою, з яких п'ять фіналістів були відібрані до фінального раунду. В результаті перемоги в конкурсі шифр Rijndeal тепер відомий як AES [9].

Однак шифр Rijndael не точно такий, як стандартизована версія AES. Шифр Rijndael підтримує три різні розміри блоків: 128, 192 та 256 біт. Однак стандартний шифр AES визначає розмір блоку лише 128 біт. Шифр AES підтримує три різні довжини ключів, необхідні Національним інститутом стандартів і технологій: 128, 192 і 256 біт.

Короткий огляд структури шифру AES розміром блоку 128 біт такий. Структуру функції шифрування AES можна розглядати як мережу заміщення-перестановки. Структура такої мережі проілюстрована на Рис2.1. 128-бітні круглі клавіші виробляються за розкладом ключів з головного ключа. Круглі клавіші змішуються у функції круглої форми із станом блоку за допомогою XOR. Раундова функція виконує фазу заміщення, передаючи шістнадцять байт стану блоку через 8-бітну функцію S-box. S-поле AES виконує мультиплікативне обернене, зокрема, кінцеве поле GF (28) перетворення у вигляді множення бітової матриці. Як результат, функція S-box - це бієкція, необхідна для побудови мережі перестановки. Зворотний блок S для

дешфрування побудований з оберненої трансформації та тієї ж мультиплікативної інверсії в GF (28). Фаза заміщення дешифрування, яка виконує шістнадцять паралельних зворотних операцій S-box, називається InvSubBytes [10].

Фаза перестановки AES розділена на дві різні фази - ShiftRows та MixColumn. Для цих фаз стан блоку можна розглядати як 4 × 4-байтну матрицю. ShiftRows обертає значення в матриці стану на різні величини вздовж рядків. Перший ряд не обертається (або можна сказати, що він обертається на нульові місця), другий ряд - на одне місце, третій - на два та останній ряд - на три місця. Фаза MixColumn виконує множення матриць у конкретному кінцевому полі GF (28) з кожним стовпцем окремо як вхід і вихід. Обидві ці операції можуть бути змінені, і ці звороти, які використовуються при дешифруванні, називаються InvShiftRows та InvMixColumns. Схему роботи AES алгоритму зображено на рисунку 2.1.

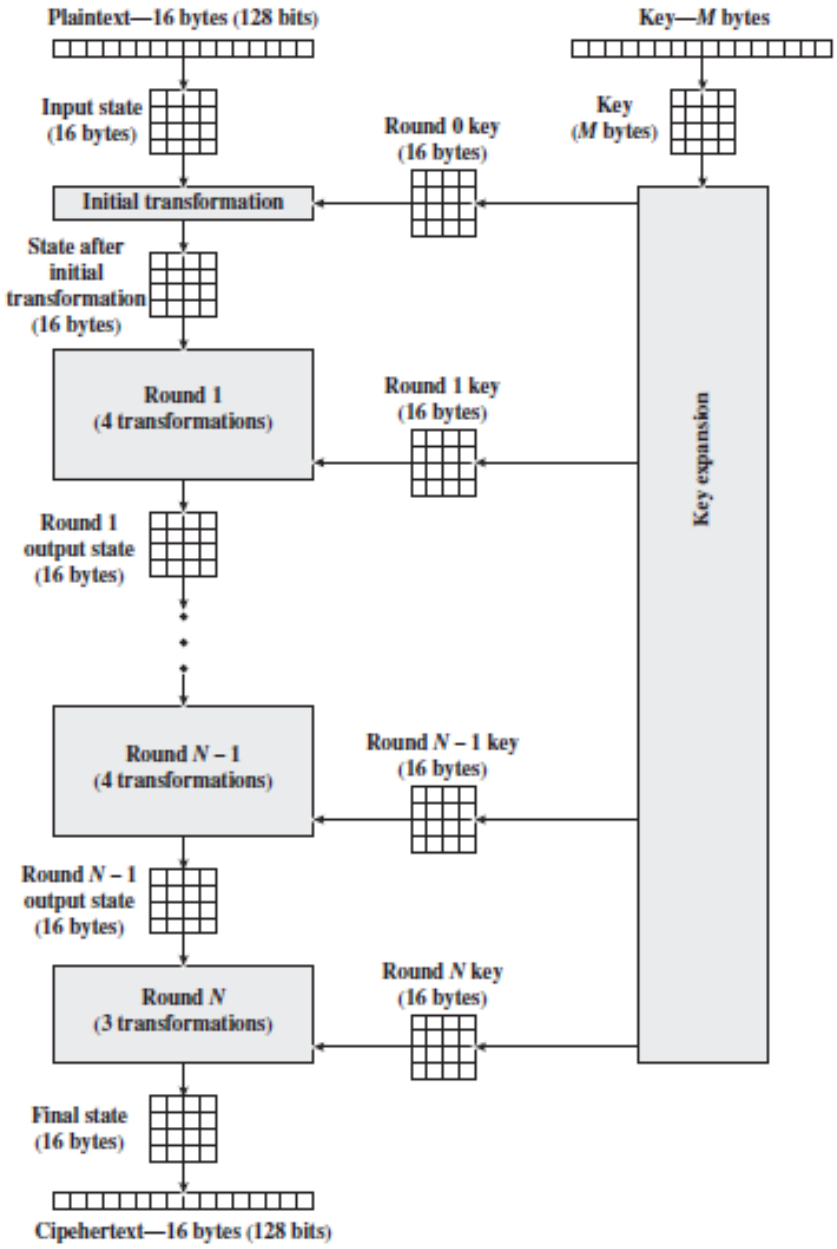


Рисунок 2.1 – Схема роботи AES алгоритму.

Процес шифрування AES починається з початкової трансформації, що є додатковою операцією AddRoundKey. Дешифрування можна виконати, змінивши порядок [11].

В даний час криптографія AES широко прийнята і підтримується як в апаратному, так і в програмному забезпеченні. До теперішнього часу жодної практичної криптоаналітичної атаки проти AES не виявлено. Крім того, AES має вбудовану гнучкість довжини ключів, що дозволяє певним чином захищати майбутнє від прогресу у можливості виконувати вичерпні пошуки ключів.

Однак, як і для DES, безпека AES гарантується лише в тому випадку, якщо вона правильно впроваджена та використовується гарне управління ключами.

2.2 Blowfish

Blowfish був розроблений в 1994 році, і це був один з перших шифрів, який був опублікований як патентний. В результаті Blowfish набув популярності і його використовують різні продукти. Схему роботи Blowfish алгоритму зображено на рисунку 2.2.

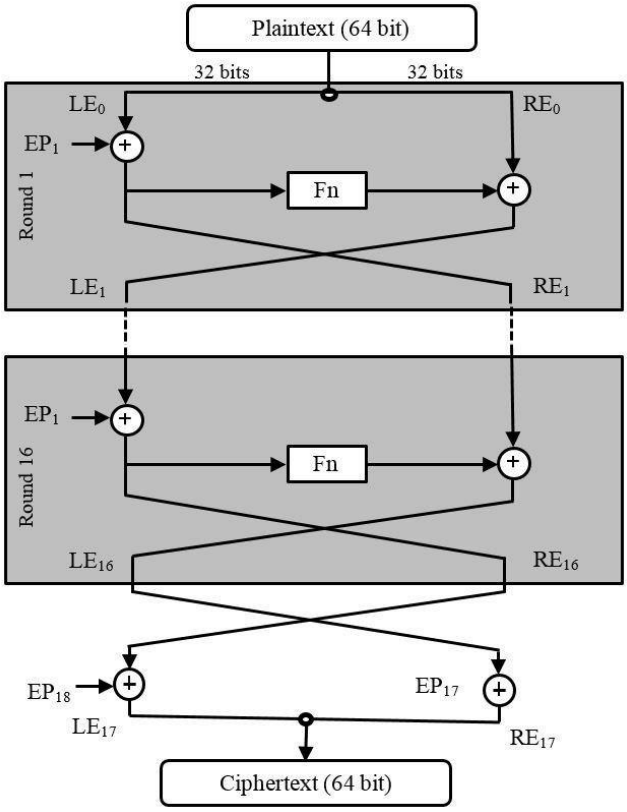


Рисунок 2.2 – Схема роботи Blowfish алгоритму.

Blowfish – це шифр Фейстеля з розміром блоку 64 біт. Підтримувана довжина головних ключів для Blowfish становить від 32 до 448 біт. Ключовий графік Blowfish виконується перед шифруванням, і він генерує чотири 8 × 32-бітові масиви, які використовуються як чотири S-вікна за допомогою функції рауда. Це робить S-скриньки безпосередньо залежними від головного ключа. Додаткові вісімнадцять 32-бітних раундові масиви також генеруються за допомогою розкладу ключів.

Дешифрування може бути виконано за допомогою функції шифрування лише шляхом зміни порядку чергових клавіш. Таким чином, реалізація Blowfish

в цьому сенсі проста, оскільки реалізація функції шифрування також дає реалізацію функції дешифрування.

Розклад ключів Blowfish починається з ініціалізації P-масиву та S-скриньки зі значеннями, отриманими з шістнадцяткових цифр pi, які не містять очевидного шаблону (див. Нічого в номері мого рукава). Після цього секретний ключ, байт за байтом, при необхідності перемикає ключ, XORed з усіма P-записами в порядку. Потім 64-бітний блок з усіма нулями шифрується з алгоритмом. Отриманий шифротекст замінює P1 і P2. Потім той самий шифротекст знову зашифровується за допомогою нових підрозділів, і новий шифротекст замінює P3 та P4. Це продовжується, замінюючи весь P-масив та всі записи S-box. Загалом алгоритм шифрування Blowfish запуститься 521 раз для генерації всіх підрозділів - обробляється близько 4 КБ даних.

Оскільки масив P має 576 біт, а ключові байти XOR через усі ці 576 біт під час ініціалізації, багато реалізацій підтримують розміри ключів до 576 біт. Причиною цього є розбіжність між оригінальним описом Blowfish, в якому використовується 448-бітний ключ, та його контрольною реалізацією, в якій використовується 576-бітний ключ. Тестові вектори для перевірки сторонніх реалізацій також були виготовлені з 576-бітними ключами. На запитання, яка версія Blowfish є правильною, Брюс Шнайер відповів: "Тестові вектори повинні використовуватися для визначення єдиного справжнього Blowfish".

Інша думка полягає в тому, що обмеження 448 біт є для того, щоб забезпечити, що кожен біт кожного підрозділу залежить від кожного біта ключа, оскільки останні чотири значення P-масиву не впливають на кожен біт шифротексту. Цей пункт слід враховувати для реалізацій з різною кількістю раундів, оскільки, хоча це збільшує захист від вичерпної атаки, він послаблює безпеку, гарантовану алгоритмом. І зважаючи на повільну ініціалізацію шифру з кожною зміною клавіші, йому надається природний захист від нападів грубої сили, що насправді не виправдовує розміри ключів довше 448 біт.

Blowfish - це швидкий блоковий шифр, за винятком випадків зміни клавіш. Кожен новий ключ вимагає попередньої обробки, еквівалентної шифруванню близько 4 кілобайт тексту, що дуже повільно порівняно з іншими

блоковими шифрами. Це запобігає його використанню в одних програмах, але не є проблемою в інших.

В одному додатку повільна зміна клавіш Blowfish - це фактично користь: метод хешування паролів (криптовалюта $ 2, тобто bcrypt), що використовується у OpenBSD, використовує алгоритм, отриманий від Blowfish, який використовує графік повільних клавіш; ідея полягає в тому, що необхідні додаткові обчислювальні зусилля захищають від атак на словники.

Blowfish має слід пам’яті трохи більше 4 кілобайт оперативної пам’яті. Це обмеження не є проблемою навіть для старих настільних і портативних комп'ютерів, хоча і не дозволяє використовувати в найменших вбудованих системах, таких як ранні смарт-карти.

Blowfish був однією з перших захищених блокових шифрів, на які не поширюються ніякі патенти, і тому вільно доступний для будь-якого користувача. Ця вигода сприяла його популярності в криптографічному програмному забезпеченні.

2.3 Twofish

Twofish - шифр Фейстеля і був одним із п’яти фіналістів конкурсу AES. Як і AES, розмір блоку Twofish становить 128 біт, він підтримує три довжини ключа 128, 192 та 256 біт. Вхідний простий текст до функцій шифрування розділений на дві половини, як і у типових шифрів Фейстеля. Більше того, половини розділені на два 32-бітні значення, на яких виконується більшість операцій.

Шифр Twofish налаштовує свої S-скриньки за допомогою головного ключа, і тому практичні реалізації програмного забезпечення обчислюють таблиці перед шифруванням або дешифруванням. Схему роботи Twofish алгоритму зображено на рисунку 2.3.

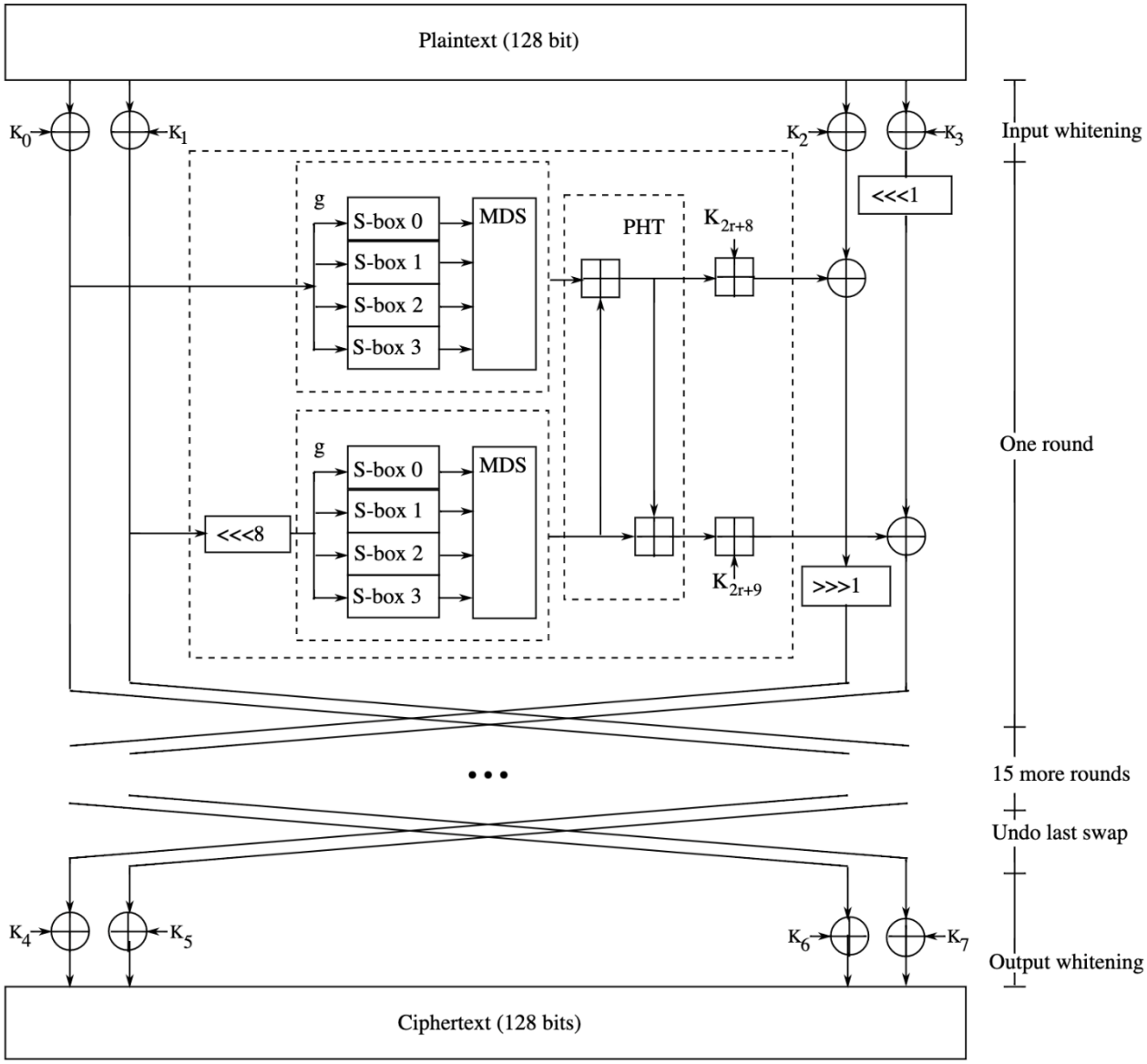


Рисунок 2.3 – Схема роботи Twofish алгоритму

Кругла функція F, g-функція - це частина, в якій виконуються функції S-box. Оскільки g-функція використовує лише 32-бітний вхід, а вхід до F-функції - це стан 64-бітного блоку, g-функція виконується двічі. Виходи обох g-функцій потім змішуються у PHT-функції, і кругла клавіша змішується зі станом, використовуючи 32-бітове додавання цілих чисел.

Результат F-функції виводиться у вигляді двох 32-бітних значень, які змішуються зі станом правого. 32-бітні значення правого стану додатково обертаються однобітним лівим і правим. Ці обертання були введені для ускладнення криптоаналізу, але вони мають недоліки. Реалізація програмного забезпечення сповільнюється приблизно на 5% завдяки впровадженню однорозрядних обертань. Ці обертання також порушують типову структуру

Фейстеля, так що функцію дешифрування неможливо побудувати з функції шифрування, просто змінивши порядок круглих клавіш.

Практично всі алгоритми шифрування мають певну процедуру налаштування ключів: спосіб взяти ключ і зробити круглі підрозділи, які алгоритм використовує. Twofish потрібно взяти ключ і зробити залежні від ключа S-коробки та круглі підрозділи. Blowfish, якому потрібно було зробити те саме, повільно налаштовував ключ, займаючи до 521 шифрування. Twofish набагато швидше; його налаштування ключів може бути таким же швидким, як 1,5 шифрування Blowfish.

Twofish має найрізноманітніші варіанти. Ви можете зайняти більше часу для налаштування ключа, і шифрування працює швидше; це має сенс для шифрування великої кількості простого тексту тим самим ключем. Ви можете швидко встановити ключ, а шифрування повільніше; це має сенс для шифрування серії коротких блоків із швидко змінюючими клавішами.

2.4 Serpent

Розробники алгоритму були зосереджені на отриманні максимально можливого рівня безпеки від будь-якого виду атак. Вони почали вивчати блокове шифрування на доступному обладнання, яке було виявлено в той період, і для свого алгоритму вирішили використати вдвічі більше «раундів», необхідних для блокування відомих атак. Метою було створення та алгоритм, який міг би гарантувати життєвий цикл 100 років. Вони змогли отримати алгоритм, який був у два рази швидшим, ніж DES.

Перемога AES над Serpent під час процесу відбору стандартизації, проведеного NIST (National Inctitute of Standards and Technology), викликала справжнє розчарування у розробників Serpent до того, що в березні 2000 року вони написали "The case of Serpent" (що в перекладі – “Справа про змію”), повністю розкривши всі помилки оцінювання, які, на їх думку, допущено NIST.

Перше сповіщення було про процесор, який використовував NIST для обчислення першого раунду (перша фаза шифрування).

За словами розробників Serpent, використовуваний процесор (Pentium 200 МГц) виявився недостатньо потужним у порівнянні зі зростаючими можливостями обчислення процесорів 21 століття.

Крім того, було сказано, що під час процесу відбору найважливішими критеріями оцінки алгоритму був рівень безпеки, який використовувався. Девіз процесу відбору полягав у тому, що алгоритм виграшу буде названий "Алгоритм 21 століття", в основному для встановлення того, що створений алгоритм повинен був тривати в часі, отже, математично безпечнішим.

Serpent шифр був одним із п’яти фіналістів конкурсу AES. Як і всі шифри в конкурсі AES, Serpent має 128-розрядний розмір блоку і підтримує розміри ключів 128, 192 і 256 біт. Схему роботи Serpent алгоритму зображено на рисунку 2.4.

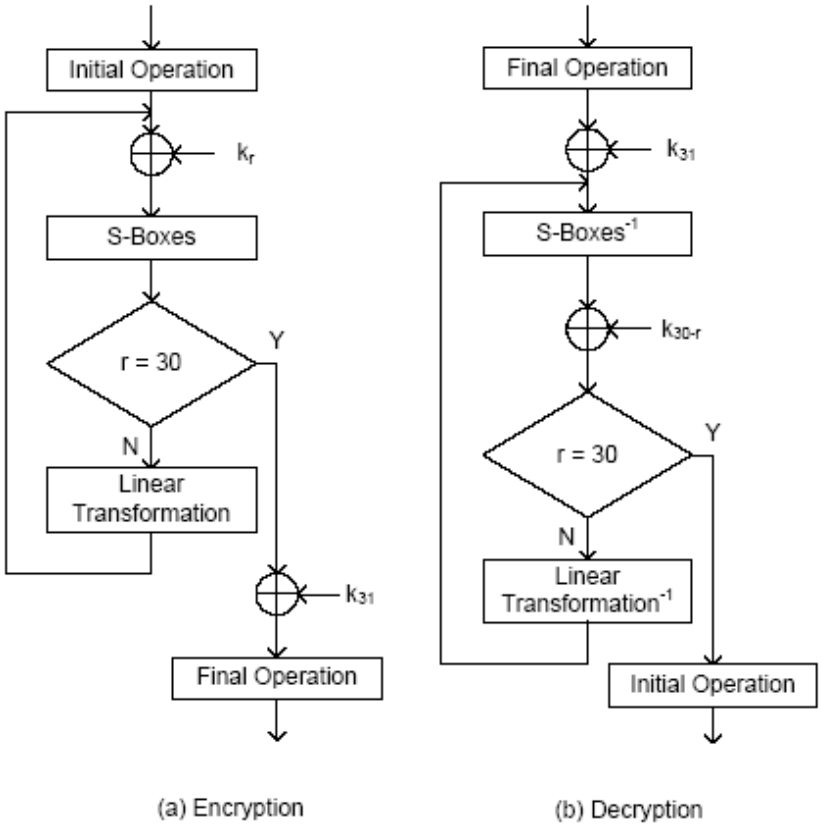


Рисунок 2.4 – Схема роботи Serpent алгоритму.

Функція шифрування Serpent використовує структуру мережі підстановки-перестановки. Serpent створений з більш сильним акцентом на безпеку, тоді як AES робить акцент на ефективності. Крім того, оскільки цей алгоритм розроблений на основі оптимізації розрізання бітів, вхід і вихід для

функції шифрування не повинні перетворюватися на бітову нарізану форму. Оскільки один і той же S-блок застосовується 32 рази паралельно у кожному раунді, фаза заміщення може бути реалізована в бітовому нарізанні, обробляючи стан блоку як 32 окремі 4-бітні блоки. Крім того, фаза перестановки розроблена таким чином, щоб вона могла бути ефективно реалізована для реалізації бітових зрізів.

Ми не знаємо, як розвиватимуться методи криптовалюти в майбутньому, розробники Serpent були натхнені трьома наступними критеріями, що дало їм мету створити алгоритм, який би тривав якомога довше з точки зору безпеки:

- код у блоках повинен бути максимально простим та легким для аналізу. Наприклад, коли DES був створений наприкінці 80-х, надана документація була настільки складною, що ніхто не намагався її атакувати. Як тільки алгоритм був вимушений лінійною та диференційованою атакою, зрозуміло, що для розуміння логіки, що стоїть за ним, потрібно лише 50 хвилин.
- блок шифрування повинен мати кілька фаз "кругового" шифрування стосовно того, що потрібно для блокування нинішніх атак через те, що обчислювальні можливості постійно розвиваються і зазвичай логіка атаки безпосередньо пов'язана з тим, що збільшується кількість раундів - це те, що призводить до того, що атаки не мають успіху.
- крім того, код у блоках повинен використовувати лише добре відомі математичні операції, що використовуються в криптографії. З цієї причини розробники Serpent використовують мережу S-P (заміна - перестановка), що використовується вже більше чверті століття, як базу для сучасних криптографічних систем, отже, зводячи до мінімуму можливість виявлення нових прийомів атаки.

Складні алгоритми важко правильно використовувати, натомість Serpent дуже простий і його можна оптимізувати за допомогою мов програмування, таких як C, що дозволяє розробникам значно полегшити роботу на етапі складання рутини.

2.5 Шифрування даних Standard DES

Стандарт шифрування даних (DES) був розроблений в 1974 році кандидатом IBM на основі попереднього алгоритму, шифру Люцифера Хорста Фейстеля. Цей алгоритм був розроблений для потреб комп'ютерної безпеки уряду США. Пізніше було зафіксовано багато атак, які зіткнулися з декількома слабкими сторонами алгоритму DES. Атака грубої сили стала головною причиною відмови алгоритму DES, оскільки був фіксований розмір ключа 56 біт (+ 8 біт парності).

Алгоритм DES – найпопулярніший алгоритм безпеки. Це симетричний алгоритм, який означає, що ті ж ключі використовуються для шифрування та дешифрування даних. Довжина ключа - 8 байт (64 біт). Отже, алгоритм DES використовує 8-байтний ключ, але 1 байт (8 біт) для перевірки парності. Це алгоритм блочного шифрування - тому розмір блоку даних алгоритму DES становить 64 біт. Для шифрування та дешифрування даних алгоритм DES використовує структуру Feistel. Хоча розмір блоку даних становить 64 біт, кількість раундів складе 16 раундів. Отже, він використовуватиме різні підрозділи для кожного раунду. тому кількість підрозділів буде 16. Схему роботи DES алгоритму зображено на рисунку 2.5.

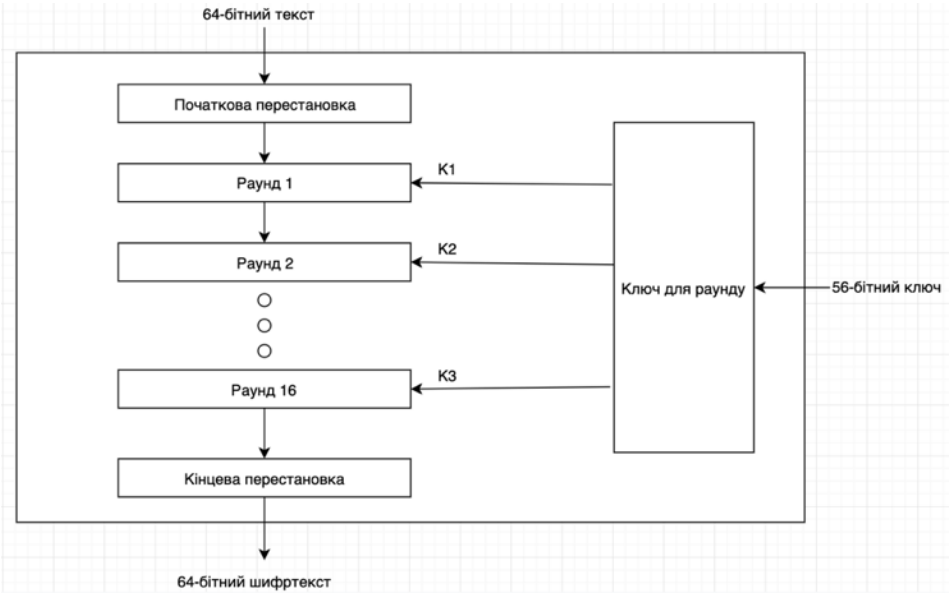


Рисунок 2.5 – Схема роботи DES алгоритму.

2.6. Triple DES

Швидкість вичерпних ключових пошуків щодо DES після 1990 року почала викликати дискомфорт у користувачів DES. Однак користувачі не хотіли замінювати DES, оскільки для зміни алгоритмів шифрування, які широко прийняті та вбудовані у великі архітектури безпеки, потрібно величезна кількість часу та коштів.

Прагматичний підхід полягав не в тому, щоб повністю відмовитися від DES, а змінити спосіб використання DES. Це призвело до модифікованих схем Triple DES (іноді відомих як 3DES). Між іншим, є два варіанти Triple DES, відомі як 3-ключовий Triple DES (3TDES) та 2-key-Triple DES (2TDES).

Потрійний алгоритм DES такий же, як і алгоритм DES, за винятком того, що ми застосовуємо його три рази. Отже, щоб зрозуміти потрійний DES, нам потрібно зрозуміти, як алгоритм DES використовується для шифрування даних та генерації ключа. DES виконує початкову перестановку на 64-бітовому блоці даних. Потім він розбиває його на дві частини з назвою L і R, кожен 32-бітний підблок. Тоді шифрування блоку повідомлення відбувається в 16 раундів. З клавіші введення генерується шістнадцять 48-бітних клавіш, по одній на кожен раунд. Права половина розширена з 32 до 48 біт. Результат поєднується з підключенням для цього раунду за допомогою операції XOR.

За допомогою S-коробок 48 отриманих бітів потім знову перетворюються на 32 біти, які згодом знову перестановляються за допомогою ще однієї фіксованої таблиці. Права половина, яка до цього часу ретельно переміщена, тепер

поєднується з лівою половиною за допомогою операції XOR. У наступному раунді ця комбінація використовується як нова ліва половина. Цей процес проводиться протягом усіх 16 раундів. Функція f на наступному малюнку робить все відображення у всіх раундах.

2.7 RSA

Алгоритм RSA характеризується досконалою безпекою, простим управлінням ключами, він є простим у виконанні та розумінні. Однак в

алгоритмі шифрування RSA модульна потужність серйозно позначається на продуктивності алгоритму і може стати вузьким місцем, що обмежує його більш широке застосуванням. Тому швидка реалізація алгоритму шифрування RSA (включаючи оптимізацію алгоритму та процес оптимізації) була зосереджена на дослідженнях з багатоядерними технологіями та паралельним розвитком технологій, обчислювальна потужність комп'ютерних систем була величезним оновленням. Використовуючи OpenMP, Pthreads та інші багатопотокові технології, можна використовувати більше обчислювальних механізмів всередині процесора, підвищувати ефективності процедур та за допомогою повідомлень MPI (Message Passing Interface), що передає паралельний інтерфейс, можна активувати кілька вузлів для спільного завершення обчислень для того, щоб скоротити операцію.

Але з появою технології CUDA (Compute Unified Device Architecture), стало можливим виконувати обчислення загального призначення на GPU (Graphics Processing Unit). Платформа CUDA Nvidia приваблює програмістів із програмою, яка надає тисячі потокових процесорів, доступних в GPU здатних забезпечити дивовижне паралельне прискорення в одному вузлі. З алгоритмом RSA і містичною силою CUDA, постає мета вивчити доцільність роботи паралельної програми алгоритму RSA і використати JCUDA для здійснення паралелізації.

Принцип RSA алгоритму: у криптографії RSA є алгоритмом відкритого ключа. Це алгоритм, який, підходить для підписання, а також шифрування, і був одним із перших чудових прогресів криптографії відкритого ключа. RSA широко використовується в Україні в електронних комерційних протоколах.
RSA включає відкритий та приватний ключ. Відкритий ключ може бути відомий всім і використовується для шифрування повідомлень. Повідомлення, зашифровані відкритим ключем, можуть бути розшифровані лише за допомогою приватного ключа. Ключі для RSA алгоритму формуються наступним чином:

Крок 1. Вибираються два чіткі прості числа p і q. В цілях безпеки повинні бути цілі p і q, які вибираються випадковим чином і мають бути

взаємнопростими. Прості цілі числа можна ефективно знайти використовуючи тест на перевірку чи є два числа простими [12];

Крок 2. Обчислити n = pq, n використовується як модуль як для публічного, так і для приватного ключа;

Крок 3. Обчислити φ (pq) = (p - 1) (q - 1). (φ - функція Ейлера);

Крок 4. Вибрати ціле число e таким, що 1 <e <φ (pq), де e і φ(pq) не мають дільника, крім 1 (тобто, e і φ(pq) є простими або відносно простими), e представляється як показник відкритого ключа з короткою довжиною і невеликою вагою Хеммінга призводить до більш ефективного шифрування. Однак малі значення для e (наприклад, e = 3) менше захищають в деяких налаштуваннях;

Крок 5. Визначити d (приватний ключ), який задовольняє відношення de(mod φ(pq)) = 1. Це часто обчислюється за допомогою розширеного алгоритму Евкліда. Де d зберігається як показник приватного ключа.

Відкритий ключ складається з модуля n та публічної експоненти шифрування e. Приватний ключ складається приватного показника дешифрування d, який слід зберігати в секреті. В алгоритмі RSA простий текст розподілений на пакети як одиниця шифрування, і шифро-текст також розділений на пакети як одиниці, що підлягають дешифруванню, в якій кожен розмір пакета - двійкове значення, яке менше числа n. Тобто, розмір пакета повинен бути меншим або рівним log2(n);

На практиці розмір пакету = k біт, у яких, $2k < n < 2k+1$.

Припустимо приватний ключ - (d, n), відкритий ключ (e, n), для простого тексту пакету M і шифро-тексту пакету C, форма шифрування і дешифрування: $C = Me \bmod n$, $M = Cd \bmod n$.

Наприклад, public key (e, n) - (13, 2537), private key (d, n) - (937, 2357), припустимо, що відкритий текст M, який потрібно надіслати, є: public key encryptions.

По-перше, відправник розділяє відкритий текст на пакети: pu bl ic ke ye nc ry pt io ns

Припустимо, що a - 00, b - 01, c - 02, … , y - 24, z - 25, і потім шифруємо пакети прямого тексту, отримуючи результат: 1520 0111 0802 1004 2404 1302 1724 1519 0814 1418

Використовуючи перетворення шифрування C = Me (mod n), отримаємо шифро-текст наступним чином: 0095 1648 1410 1299 1365 1379 2333 2132 1751 1289. Відправник передає шифро-текст на одержувач, і приймач з використанням дешифрувального перетворення M = Cd (mod n), отримаємо простий текст наступним чином: 1520 0111 0802 1004 2404 1302 1724 1519 0814 1418. Перетворимо простий текст в алфавіт, який є оригінальним: public key encryptions.

Огляд архітектури для реалізації RSA алгоритму.

Шифрувальне навантаження: в процесі шифрування RSA немає кореляції та залежності між даними пакетів. Отже, ми можемо використовувати метод декомпозиції домену, всі дані будуть класифіковані як піддомени, кожен потік або процес обчислює піддомен. Через паралелізм даних метод [3] збільшує обчислювальну швидкість RSA. Вузьке місце процесу шифрування RSA - великий розмір даних, використовується CUDA для здійснення паралельного шифрування даних між пакетами. Ця структура програми дозволяє конкретні процеси шифрування RSA для паралельного процесу.

Огляд апаратної архітектури CUDA: кожен пристрій, сумісний із CUDA, є набором мультипроцесорів, які здатні виконувати велику кількість потоків одночасно, і це функціонує як співпроцесор основного ЦП або хост. Кожен мультипроцесор має архітектуру SIMD (single instruction, multiple data), тобто кожен процесор мультипроцесора виконується як інший потік, але всі потоки виконують ту саму інструкцію, оперуючи різними даними на основі Id потоку у будь-який заданий тактовий цикл. NVIDIA GTX 280 має тридцять мультипроцесорів з вісьмома процесорами кожен.

Хост, і пристрій підтримують власну DRAM (Dynamic Random Access Memory), згадується як пам'ять хоста і пам'ять пристрою (вбудована пам'ять). Пам'ять пристрою може бути трьох різних типів: глобальна пам'ять, постійна пам'ять і текстурна пам'ять. Всі вони можуть читати або записувати хост і є

стійкими протягом життя програми. Тим не менш, глобальні, постійні та текстурні простори пам'яті оптимізовані для різних задач пам'яті. Мультипроцесори мають вбудовану мікросхему пам'яті, яка може бути чотирьох наступних типів: регістри, спільна пам'ять, постійний кеш і кеш текстури. Апаратна модель архітектури CUDA зображена на рисунку 2.6.

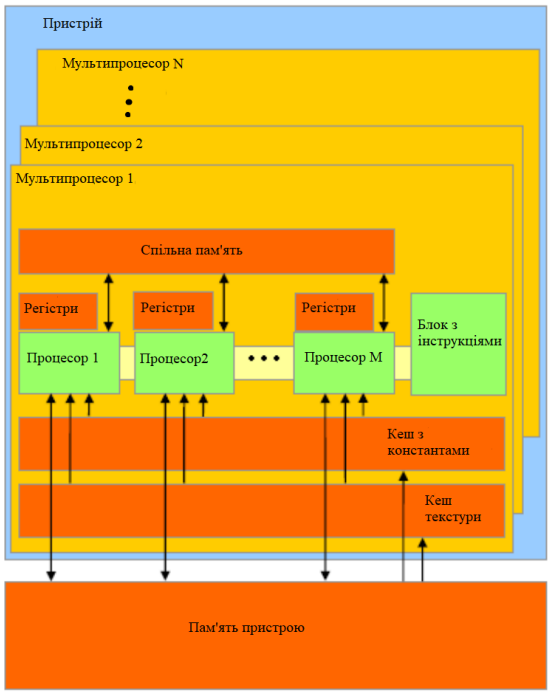


Рисунок 2.6 – Апаратна модель CUDA

Кожен процесор у багатопроцесорі має один набір локальних 32- бітових регістрів читання-запису на один процесор. Паралельний кеш даних загальної пам'яті ділиться всіма процесорами. Лише для читання постійний кеш обмінюється всіма процесорами і прискорює роботу читання з постійною пам'яттю. Кеш текстури, доступний лише для читання, він є спільним для всіх мультипроцесорів і має один набір локальних 32-розрядних регістрів читання-запису на один процесор. Паралельний кеш даних спільної пам'яті поділяється всіма процесорами. Локальні та глобальні простори пам'яті реалізуються як регіони читання-запису пам'яті пристрою та вони є не кешованими. NVIDIA GTX 280 має 16 384 регістрів та 16 Кб спільної пам'яті на мультипроцесорі.

Основним недоліком продуктивності CUDA є вартість передачі даних з пам'яті хоста в пам'ять пристрою. Усі функції CUDA потребують доступу до пам'яті пристрою та роботи з ними при передачі даних. Це означає, що перед тим як

код CUDA виконається, нам потрібно перенести всі дані на пристрій, а потім, після завершення коду, потрібно перенести його назад.

2.8 Висновки до розділу

У даному розділі було подано огляд найпоширеніших блокових криптографічних алгоритмів. З’ясоване походження алгоритмів та подана схема їх роботи. Запропоновані можливі варіанти їх використання.

3 ПРОПОНОВАНА ПАРАЛЕЛЬНА СИСТЕМА

Реалізовано паралельну систему з використанням багатоядерного процесора паралельно для підвищення продуктивності шифрування. Використовуючи паралельне шифрування файлів з бібліотеками API для виконання самого шифрування / дешифрування та бібліотеку паралельного програмування (PPL) для паралелізації в комп'ютерній системі. з генерованими фіксованими розмірами файлів для експерименту ми розробляємо, впроваджуємо, тестуємо та оцінюємо паралельну версію розглянутих алгоритмів (AES, Blowfish, Twofish, DES, Triple DES, Serpent).

3.1 Мови паралельного програмування

Паралельне програмування - це програмування мовою, яка дозволяє чітко вказати, як різні частини обчислень можуть виконуватися одночасно різними процесорами. Для розвитку паралельних програм було розроблено багато методик (бібліотеки, API, паралельні моделі та мови). Використовуючи Visual Studio для розробки програм для декількох пристроїв, які працюють на інших платформах. RTL (Runtime library) забезпечує бібліотеку програмування паралельних програм PPL (Parallel Patterns Library), надаючи системам можливість розподіляти завдання, що працюють паралельно, користуючись перевагою роботи на комп'ютерах з декількома процесорами.

Використовуючи PPL, запропоновані програми легко:

- роблять циклічні роботи швидше за допомогою підкласу (TParallel.For);
- виконують паралельно кілька завдань, використовуючи підклас (TTask) та (ITask);
- залишають процес, що працює, зосереджуючись на інших завданнях, а потім отримують результат обробки в потрібний час. Підклас (IFuture) дозволяє встановити пріоритет для запущених блоків коду та повернути результати при необхідності.

3.2. Проектування алгоритмів паралельного шифрування

Сучасні криптографічні алгоритми часто покладаються на математику для перетворення простого тексту в текст шифру. Таким чином, ці алгоритми можуть включати важке повторення математичного обчислення на сегменти даних (байти, блоки тощо). Режим єдиної програмної множини даних (SPMD) може бути найбільш релевантним режимом паралелізації криптографічних алгоритмів. Однак, загальними етапами проектування паралельних програм з послідовного коду є: аналіз послідовного алгоритму для виявлення потенційної паралелізації, проектування та реалізація паралельної версії (на цьому етапі можуть використовуватися методи прототипування), тест на виявлення помилок синхронізації, які можуть бути спричинені декількома потоками, та налаштування алгоритму для кращої продуктивності, видаливши всі вузькі місця.

Прототип – це початкова версія програмної системи, яка використовується для демонстрації концепцій, випробування варіантів проектування та, як правило, для того, щоб дізнатися більше про проблему та її можливі рішення, прототипування є звичайною діяльністю на етапі проектування систем. Він фокусується на ключових ідеях та основних вимогах і призводить до зменшення загального кодування, яке виділяється для вирішення проблеми. Крім того, прототип дозволяє досліджувати різні рішення та критичні аспекти, пов'язані з проблемою. Прототипи повинні бути швидко побудовані і швидко модифіковані, щоб отримати максимальну користь від процесу прототипування. Щоб досягти цього, мова повинна забезпечувати семантичні конструкції високого рівня, які дозволяють дизайнеру моделювати будь-яку структуру даних чи ідеї.

3.3 Паралельна реалізація

У розробці проекту повинні бути прийняті вісім правил багатопотокових паралельних алгоритмів:

Крок 1. Визначити незалежні компоненти серіалізованого алгоритму, оскільки деякі компоненти алгоритму можуть бути не паралельними через залежності між ними;

Крок 2. Для здійснення паралельності з використанням найвищого рівня паралелізму, є два підходи до визначення найвищого рівня зверху вниз і знизу вгору;

Крок 3. Плануйте масштабування на ранній стадії проектування та врахуйте збільшення кількості технологічних одиниць;

Крок 4. Використовувати бібліотеки, що є захищеними від ниток, де це можливо;

Крок 5. Використовуйте неявну потокову модель над явною моделлю, яка забезпечує необхідну функціональність;

Крок 6. Не вважайте, що компоненти алгоритму повинні виконуватися в певному порядку;

Крок 7. Використовуйте локальні змінні потоків якомога більше та забезпечуйте блокування спільних даних, щоб забезпечити синхронізацію;

Крок 8. Вмініть алгоритм, якщо це може забезпечити більше синхронізації, навіть якщо це збільшує складність алгоритму.

Паралелізація RSA алгоритму

Паралельна реалізація алгоритму RSA може виконуватись паралельно в два рівні: рівень потоку та комп'ютерний рівень, як видно на рисунку 3.1.

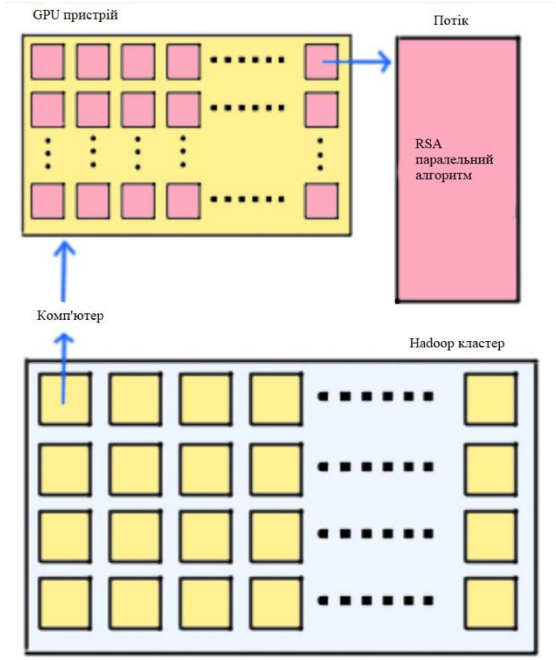


Рисунок 3.1 – Два рівні паралелізації

На рівні потоку реалізується паралельний алгоритм RSA в рамках CUDA. RSA алгоритм ділить простий текст або шифротекст на кілька пакетів

однакової довжини, операції шифрування або дешифрування будуть виконані для кожного пакету. Припустимо, що кожен пакет - це масив з тим самим номером елемента, тоді і шифрування і дешифрування може бути виконано кількома потоками, в кожен потік потрібно лише набрати елементи, які призначені саме йому та запустити ту саму функцію шифрування чи дешифрування цих елементів. Питання в тому, який потік знає, який з елементів присвоюють саме йому. Користувач CUDA може отримати індекс потоку та блоку потоку, який викликає це у функції, що працює на пристрої, щоб ми могли використовувати індекс для керування зміщенням елемента та присвоєнню елементу правильні індекси до певних ниток (потоків). У цьому рівні CUDA мультипотокова модель програмування значно покращить швидкість алгоритму RSA.

Якщо простий текст потрібно зашифрувати, або шифро-текст необхідно розшифрувати, а його довжина занадто велика, ми можемо використати розподілену файлову систему для розповсюдження тексту до кластерної системи: кожен вузол кластерної системи буде запускати цей паралельний алгоритм RSA. У цьому середовищі кластера кожен вузол отримує частину простого тексту або шифро-тексту, а потім розділяє частину на кілька рядків (просто запустіть функцію алгоритму RSA в декількох потоках).

Пристрій GPU реалізований мовою C++. Тож нам потрібен метод увімкнення перехресного дзвінка між мовою Java та C++. Java Native Interface (JNI) [13] можна використовувати як міст, але це підхід важкий і не продуктивний. Тож я обрав JCUDA [14] як рішення цієї проблеми. Модель JCUDA [15] призначена для зручності програмістів, вона надає інтерфейс для виклику ядер CUDA з кодом Java, особливо це зручно для програмістів знайомих з синтаксисом Java та CUDA.

3.4. Ефективність Бенчмаркінгу

Вимірювання продуктивності - це дуже важливий крок, який дозволяє виявити вузькі місця в алгоритмі та намагатися усунути їх з метою підвищення загальної продуктивності. Крім того, ми можемо визначити кращий рівень паралелізму або рівень точності алгоритму та паралельного середовища.

Дізнайтеся більше про коефіцієнт масштабованості в алгоритмі при збільшенні кількості одиниць обробки, які працюють разом для вирішення проблеми [16].

У цьому дослідженні ми використовуємо два фактори для оцінки продуктивності: коефіцієнт швидкості та коефіцієнт ефективності, як описано нижче.

Прискорення. Прискорення паралельного алгоритму - це відношення між часом послідовного виконання та часом виконання паралельної версії:

$$\boldsymbol{S = \frac{T_{seq}}{T_{par}}}$$

Де значення $Tseq$ - час виконання послідовної версії, а $Tpar$ - час виконання паралельної версії. Однак значення прискорення може бути представлено, наприклад, як кратне 2x, що означає, що швидкість буде подвоєна. Швидкість роботи слід змінювати, змінюючи кількість одиниць обробки; це дасть відчуття масштабованості алгоритму шляхом вимірювання ступеня зміни швидкості при зміні кількості одиниць обробки [17].

Ефективність. Ефективність підказує дизайнеру, наскільки добре використовуються обчислювальні ресурси. Ефективність можна виміряти, поділивши значення Speedup на кількість процесорних одиниць.

$$\textbf{Efficiency} = \frac{Speedup}{P}$$

Кількість одиниць обробки P є ключовим фактором рівняння ефективності; крім того, на ефективність деяких програм може негативно вплинути, коли кількість процесорних одиниць збільшується. Хоча зменшення кількості процесорних одиниць може підвищити ефективність, головне питання залишається "чому паралельна програма не може отримати максимальну користь від процесорних блоків?" Крім того, збільшення кількості потоків та призначення більше одного потоку на один процесорний блок може дещо поліпшити виконання програми. Однак значення швидкості та ефективності можуть давати підказки про рівень точності для найкращого виконання [18].

3.5 Висновки до розділу

У даному розділі було проаналізовано паралельне проектування для програмування криптографічних алгоритмів. Та подана Ефективність Бенчмаркінгу, на основі чого і буде робитися порівняльний аналіз алгоритмів.

4 РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТІВ ТА ОБГОВОРЕННЯ

У цьому розділі представлені результати паралельних алгоритмів шифрування. Вводяться створені файли з різними розмірами на однакових виділених комп'ютерах з різними процесорами. Вводимо аналіз та обговорення отриманих результатів для паралельної та послідовної реалізації.

Алгоритми, які потрібно паралелізувати: AES, Blowfish, Twofish, DES, 3DES, Serpenter – це найсильніші алгоритми та найбільш часто використовувані в даний час. Отже, ці алгоритми є дуже сильними кандидатами для реалізації та паралелізації. Отже, через обмеження в часі інші алгоритми шифрування не реалізуються або паралелізуються [19].

4.1 Експериментальне встановлення

Можна порівняти ефективність різних алгоритмів, запустивши програму Parallel File Encrypt з API Бібліотеки, що використовується для виконання самого шифрування / дешифрування, та PPL для паралелізації в комп'ютерній системі.

Ця програма виконує всі реалізовані в даний час послідовні версії та записує час виконання у зовнішній файл Excel. Розміри файлів, які використовуються в експерименті, складають: 5 МБ, 10 МБ, 100 МБ, 1000 МБ, які генеруються раніше і зберігаються у зовнішніх файлах [20].

Всі послідовні та паралельні версії були виміряні на багатоядерній машині, включаючи аналіз продуктивності, виконувались під операційною системою Windows 10 з 16 ГБ оперативної пам’яті, доступними за допомогою трьох різних процесорів, як описано нижче:

- processor Intel(R) Core(TM) i5-6500U CPU @ 2.50GHz, 2592 Mhz, 2 Cores;
- processor Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz, 2601 Mhz, 4 Cores;
- processor Intel(R) Core(TM) i7-6700Q @ 3.20GHz, 3201 Mhz, 6 Cores.

Отже, програма виконувалася для порівняння продуктивності різних алгоритмів. Для отримання достовірних показань вимірювання повторюють три рази і приймають медіану.

4.2 Результати програмної реалізації

У цьому розділі буде розроблена послідовна та паралельна версія за допомогою C++ з PPL. Ця реалізяція починає працювати з вибору вхідних та вихідних файлів, потім визначається алгоритм, який буде використовуватися з ключем шифрування. Нарешті, отримується час для паралельної та послідовної реалізації. Результати порівнюватимуться із найкращими послідовними. В результаті цього Розділу буде проведено аналіз на паралельну та послідовну реалізацію [21].

По-перше, буде оцінено послідовну продуктивності. По-друге, паралельна продуктивність буде вимірюватися налаштуванням кількості процесорів та різних розмірів файлів. Ми помічаємо відмінності використання процесора між послідовними та паралельними впровадженнями, як показано в таблиці 4.1.

Таблиця 4.1 – Отримані значення часу виконання послідовної та паралельної реалізації різними ядрами процесорів у мілісекундах з чотирма різними розмірами файлів для алгоритму DES

DES Runtime	5MB	10MB	100MB	1000MB
Sequential	112	224	2251	22428
2 Core	92	192	1872	18463
4 Cores	78	89	846	8600
6 Cores	56	64	557	5592

Відмінності значень часу виконання послідовних та паралельних реалізацій різними процесорними ядрами в мілісекундах з чотирма різними розмірами файлів для алгоритму DES зображено на рисунку 4.1.

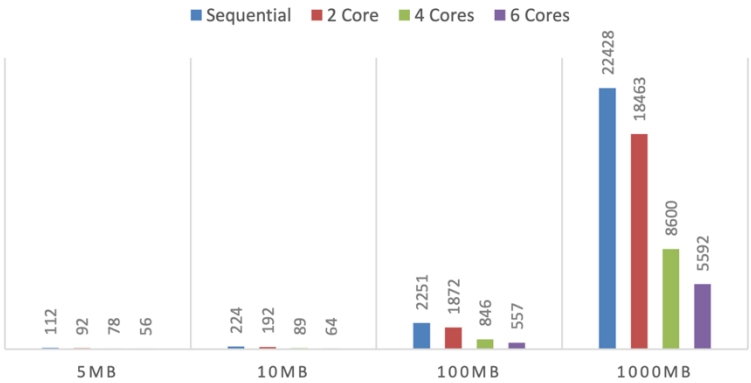


Рисунок 4.1 – Час виконання алгоритму DES

Це показує, що час паралельного шифрування краще, ніж час послідовного шифрування. Отримані значення для обчислень прискорення щодо послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму DES показано в таблиці 4.2.

Таблиця 4.2 – Прискорення алгоритму DES

DES Speedup	**5MB**	**10MB**	**100MB**	**1000MB**
Sequential	1.00	1.00	1.00	1.00
2 Core	1.22	1.17	1.20	1.21
4 Cores	1.44	2.52	2.66	2.61
6 Cores	2.00	3.50	4.04	4.01

Відмінності у значеннях прискорення послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму DES зображено на рисунку 4.2.

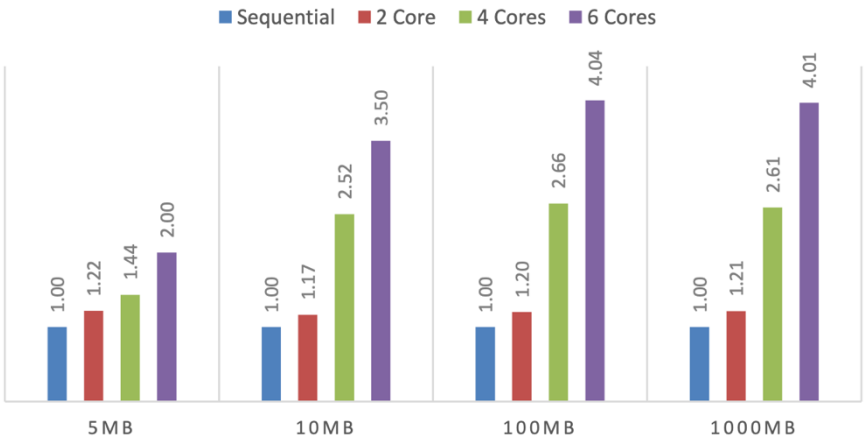


Рисунок 4.2 – Значеннях прискорення алгоритму DES

Отримані результати обчислень ефективності для послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму DES показано в таблиці 4.3.

Таблиця 4.3 – Ефективність алгоритму DES

DES Efficiency	**5MB**	**10MB**	**100MB**	**1000MB**
Sequential	1.00	1.00	1.00	1.00
2 Core	0.61	0.58	0.60	0.61
4 Cores	0.36	0.63	0.67	0.65
6 Cores	0.33	0.58	0.67	0.67

Відмінності у значеннях ефективності послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму DES зображено на рисунку 4.3.

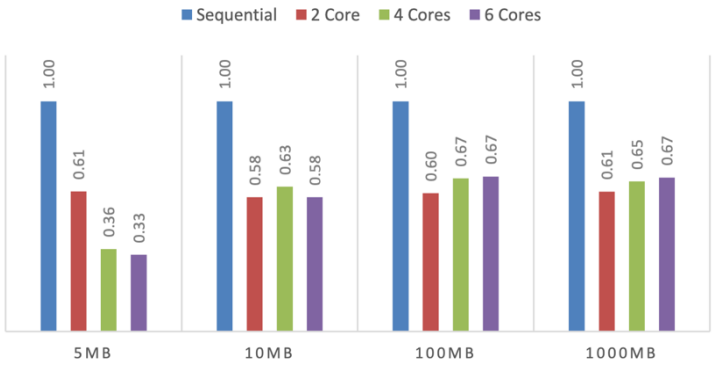


Рисунок 4.3 – Значеннях ефективності алгоритму DES

Значення часу виконання послідовної та паралельної реалізації різними процесорними ядрами в мілісекундах з чотирма різними розмірами файлів для алгоритму 3DES показано в таблиці 4.4.

Таблиця 4.4 – Час виконання алгоритму 3DES

3DES Runtime	**5MB**	**10MB**	**100MB**	**1000MB**
Sequential	298	582	5657	56987
2 Core	110	229	2148	20585
4 Cores	116	209	1910	19001
6 Cores	66	139	1099	11085

Відмінності у значеннях часу виконання послідовної та паралельної реалізації різними процесорними ядрами в мілісекундах з чотирма різними розмірами файлів для алгоритму 3DES зображено на рисунку 4.4.

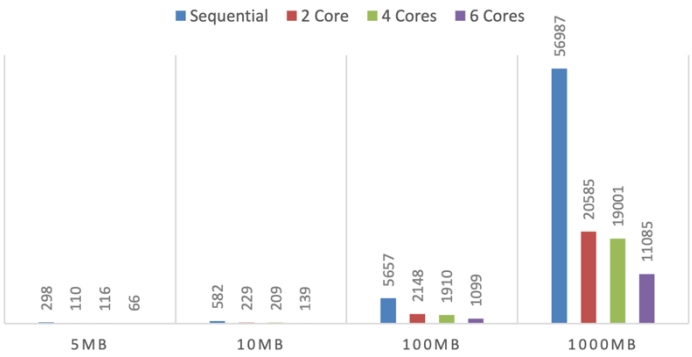


Рисунок 4.4 – Час виконання алгоритму 3DES

Отримані значення обчислень прискорення для послідовної та паралельної реалізації різними ядрами процесорів з чотирма різними розмірами файлів для алгоритму 3DES показано в таблиці 4.5.

Таблиця 4.5 – Прискорення алгоритму 3DES

3DES Speedup	5MB	10MB	100MB	1000MB
Sequential	1.00	1.00	1.00	1.00
2 Core	2.71	2.54	2.63	2.77
4 Cores	2.57	2.78	2.96	3.00
6 Cores	4.52	4.19	5.15	5.14

Відмінності у значеннях прискорення послідовних і паралельних різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму 3DES зображено на рисунку 4.5.

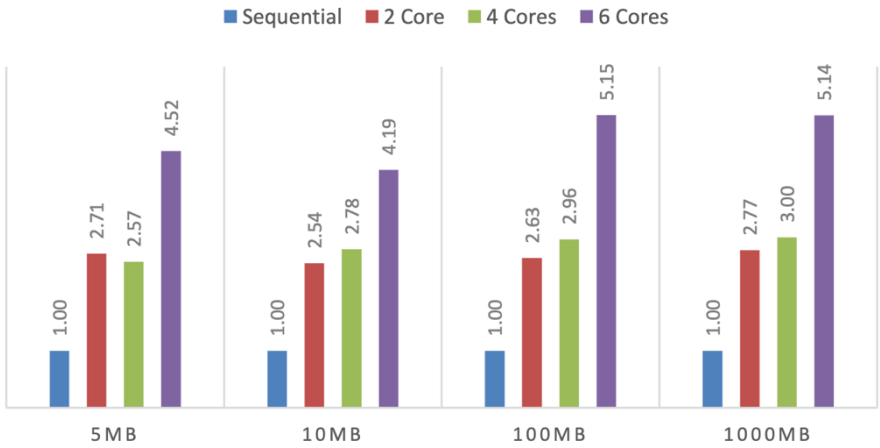


Рисунок 4.5 – Значення прискорення алгоритму 3DES

Отримані значення для обчислень ефективності послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму 3DES показано в таблиці 4.6.

Таблиця 4.6 – Ефективність алгоритму 3DES

3DES Efficiency	5MB	10MB	100MB	1000MB
Sequential	1.00	1.00	1.00	1.00
2 Core	1.35	1.27	1.32	1.38
4 Cores	0.64	0.70	0.74	0.75
6 Cores	0.75	0.70	0.86	0.86

Відмінності у значеннях ефективності послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму 3DES зображено на рисунку 4.6.

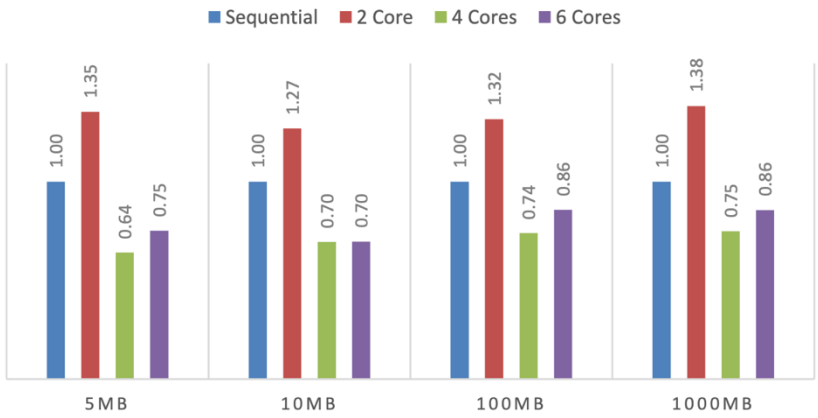


Рисунок 4.6 – Значеннях ефективності алгоритму 3DES

Значення часу виконання послідовної та паралельної реалізації різними ядрами процесора в мілісекундах з чотирма різними розмірами файлів для алгоритму AES показано в таблиці 4.7.

Таблиця 4.7 – Час виконання алгоритму AES

AES Runtime	5MB	10MB	100MB	1000MB
Sequential	77	148	1332	13597
2 Core	71	142	1445	14703
4 Cores	35	63	617	6543
6 Cores	25	47	405	5304

Відмінності значень часу виконання послідовної та паралельної реалізації різними процесорними ядрами в мілісекундах з чотирма різними розмірами файлів для алгоритму AES зображено на рисунку 4.7.

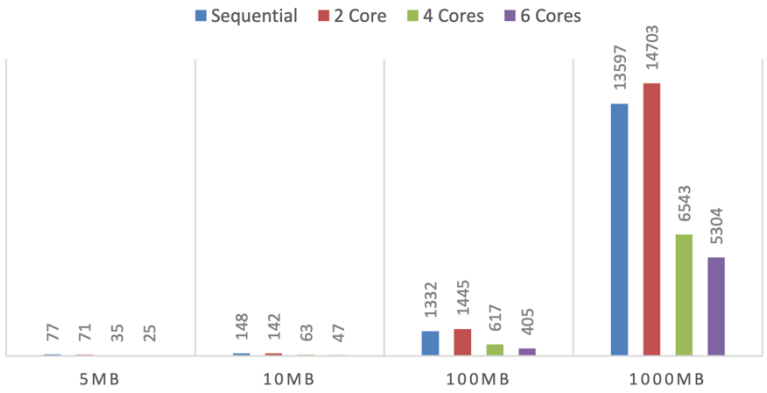


Рисунок 4.7 – Час виконання алгоритму AES

Значення прискорення послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму AES показано в таблиці 4.8.

Таблиця 4.8 – Прискорення алгоритму AES

AES Speedup	5MB	10MB	100MB	1000MB
Sequential	1.00	1.00	1.00	1.00
2 Core	1.08	1.04	0.92	0.92
4 Cores	2.20	2.35	2.16	2.08
6 Cores	3.08	3.15	3.29	2.56

Відмінності у значеннях прискорення послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму AES

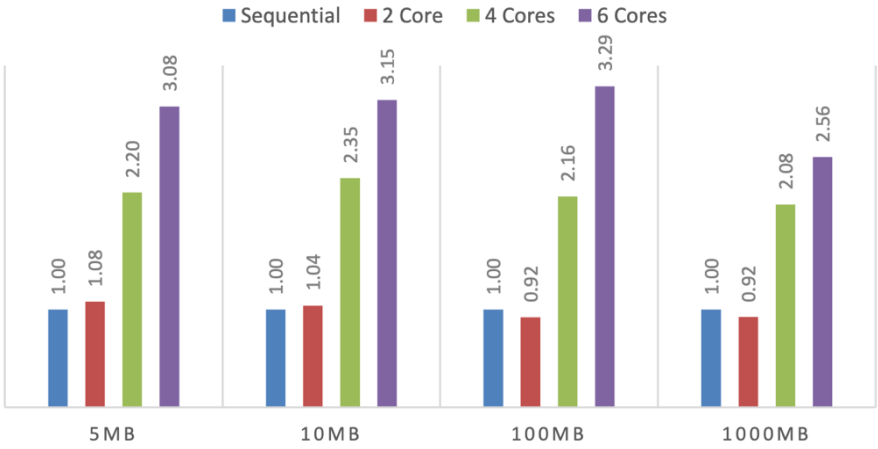


Рисунок 4.8 – Значення прискорення алгоритму AES

Значення ефективності послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму AES показано в таблиці 4.9.

Таблиця 4.9 – Ефективність алгоритму AES

AES Efficiency	5MB	10MB	100MB	1000MB
Sequential	1.00	1.00	1.00	1.00
2 Core	0.54	0.52	0.46	0.46
4 Cores	0.55	0.59	0.54	0.52
6 Cores	0.51	0.52	0.55	0.43

Відмінності у значеннях ефективності послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму AES зображено на рисунку 4.9.

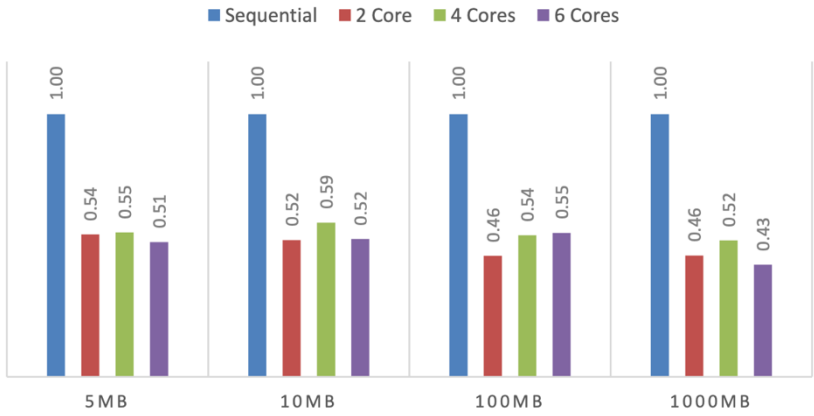


Рисунок 4.9 – Значеннях ефективності алгоритму AES

Значення часу виконання послідовної та паралельної реалізації різними ядрами процесора в мілісекундах з чотирма різними розмірами файлів для алгоритму Blowfish показано в таблиці 4.10.

Таблиця 4.10 – Час виконання алгоритму Blowfish

Blowfish Runtime	**5MB**	**10MB**	**100MB**	**1000MB**
Sequential	73	148	1456	14352
2 Core	128	248	2498	24201
4 Cores	47	74	695	6935
6 Cores	50	98	898	9623

Відмінності значень часу виконання послідовних і паралельних різними процесорними ядрами в мілісекундах з чотирма різними розмірами файлів для алгоритму Blowfish зображено на рисунку 4.10.

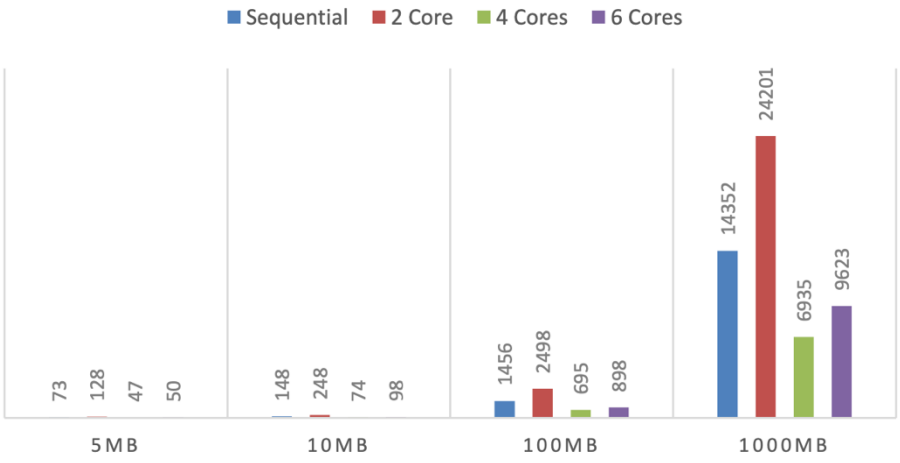


Рисунок 4.10 – Час виконання алгоритму Blowfish

Отримані значення прискорення послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму Blowfish показано в таблиці 4.11.

Таблиця 4.11 – Прискорення алгоритму Blowfish

Blowfish Speedup	5MB	10MB	100MB	1000MB
Sequential	1.00	1.00	1.00	1.00
2 Core	0.57	0.60	0.58	0.59
4 Cores	1.55	2.00	2.09	2.07
6 Cores	1.46	1.51	1.62	1.49

Відмінності у значеннях прискорення послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму Blowfish зображено на рисунку 4.11.

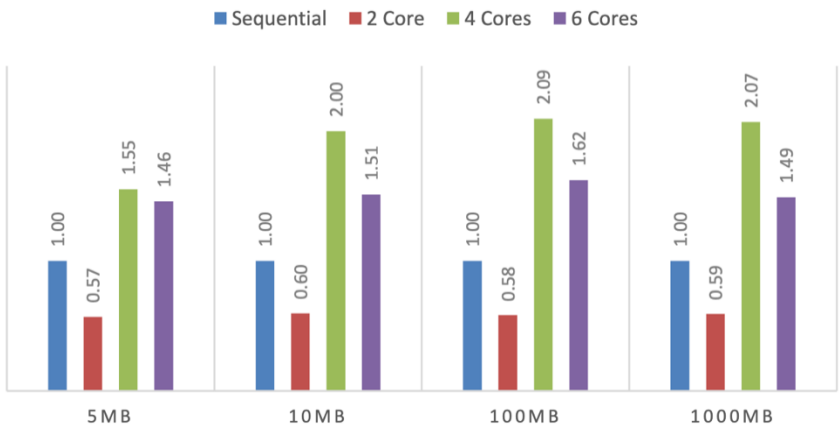


Рисунок 4.11 – Значеннях прискорення алгоритму Blowfish

Значення ефективності послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму Blowfish показано в таблиці 4.12.

Таблиця 4.12 – Ефективність алгоритму Blowfish

Blowfish Efficiency	5MB	10MB	100MB	1000MB
Sequential	1.00	1.00	1.00	1.00
2 Core	0.29	0.30	0.29	0.30
4 Cores	0.39	0.50	0.52	0.52
6 Cores	0.24	0.25	0.27	0.25

Відмінності у значеннях ефективності послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму Blowfish зображено на рисунку 4.12.

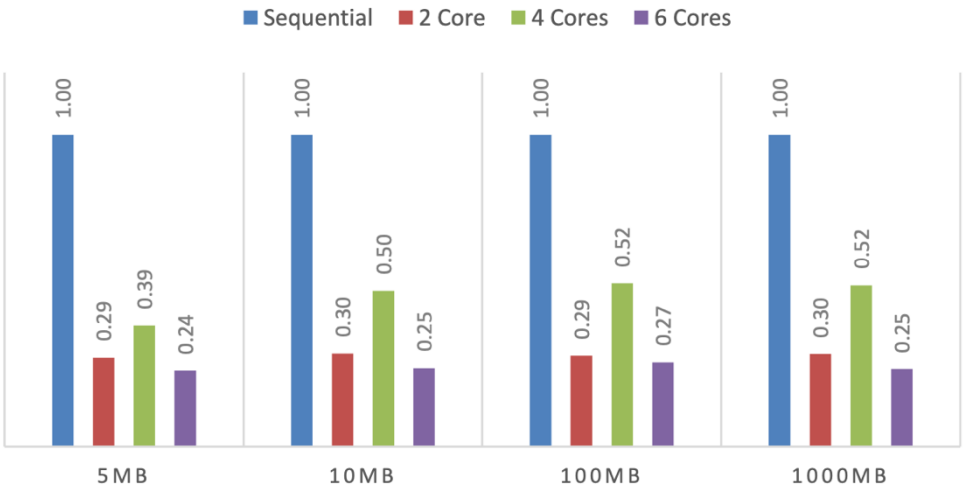


Рисунок 4.12 – Значення ефективності алгоритму Blowfish

Значення часу виконання послідовної та паралельної реалізації різними процесорними ядрами в мілісекундах з чотирма різними розмірами файлів для алгоритму Twofish показано в таблиці 4.13.

Таблиця 4.13 – Час виконання алгоритму Twofish

Twofish Runtime	**5MB**	**10MB**	**100MB**	**1000MB**
Sequential	70	131	1292	13215
2 Core	349	721	7759	78550
4 Cores	59	112	1087	11051
6 Cores	73	174	1399	16137

Відмінності значень часу виконання послідовних і паралельних різними процесорними ядрами в мілісекундах з чотирма різними розмірами файлів для алгоритму Twofish зображено на рисунку 4.13.

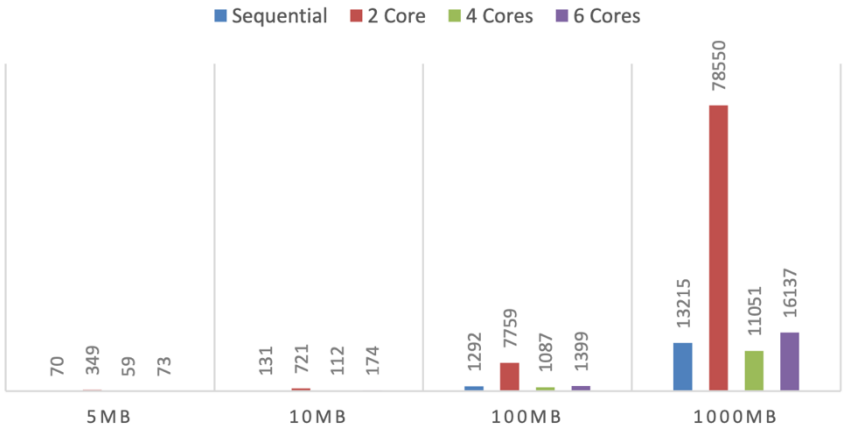


Рисунок 4.13 – Часу виконання алгоритму Twofish

Значення прискорення послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму Twofish показано в таблиці 4.14.

Таблиця 4.14 – Прискорення алгоритму Twofish

Twofish Speedup	5MB	10MB	100MB	1000MB
Sequential	1.00	1.00	1.00	1.00
2 Core	0.20	0.18	0.17	0.17
4 Cores	1.19	1.17	1.19	1.20
6 Cores	0.96	0.75	0.92	0.82

Відмінності у значеннях прискорення послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму Twofish зображено на рисунку 4.14.

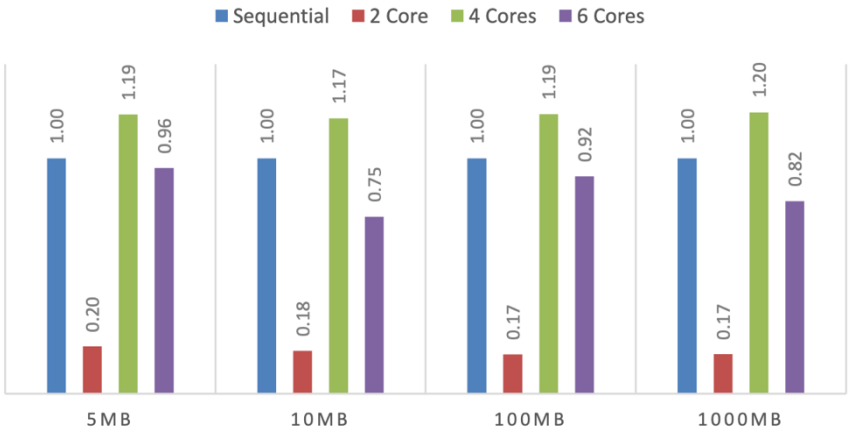


Рисунок 4.14 – Значеннях прискорення алгоритму Twofish

Значення ефективності послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму Twofish показано в таблиці 4.15.

Таблиця 4.15 – Ефективність алгоритму Twofish

Twofish Efficiency	5MB	10MB	100MB	1000MB
Sequential	1.00	1.00	1.00	1.00
2 Core	0.10	0.09	0.08	0.08
4 Cores	0.30	0.29	0.30	0.30
6 Cores	0.16	0.13	0.15	0.14

Відмінності у значеннях ефективності послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму Twofish зображено на рисунку 4.15.

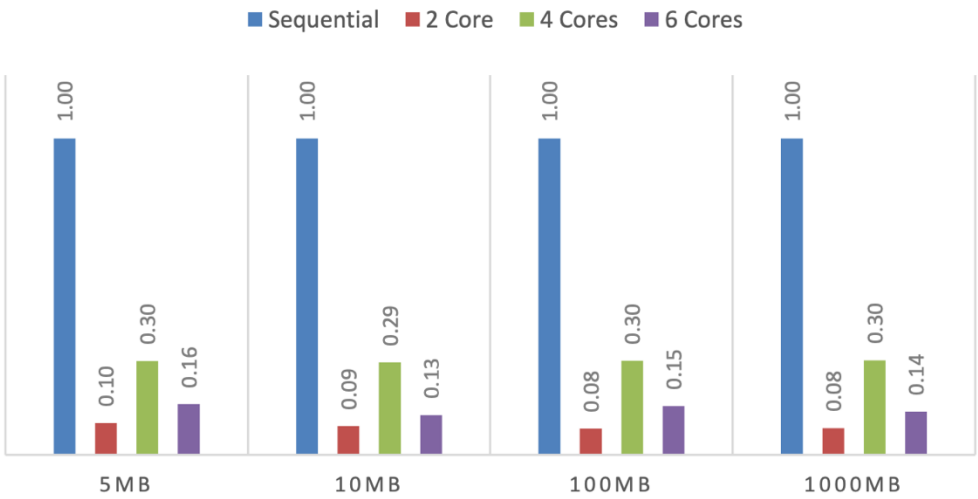


Рисунок 4.15 – Значеннях ефективності алгоритму Twofish

Показує значення часу виконання послідовної та паралельної реалізації різними процесорними ядрами в мілісекундах з чотирма різними розмірами файлів для алгоритму Serpent показано в таблиці 4.16.

Таблиця 4.16 – Час виконання алгоритму Serpent

Serpent Runtime	**5MB**	**10MB**	**100MB**	**1000MB**
Sequential	241	472	4561	45902
2 Core	104	203	1918	18619
4 Cores	93	183	1747	17628
6 Cores	57	116	944	9571

Відмінності у значеннях часу виконання послідовної та паралельної реалізації різними процесорними ядрами в мілісекундах з чотирма різними розмірами файлів для алгоритму Serpent зображено на рисунку 4.16.

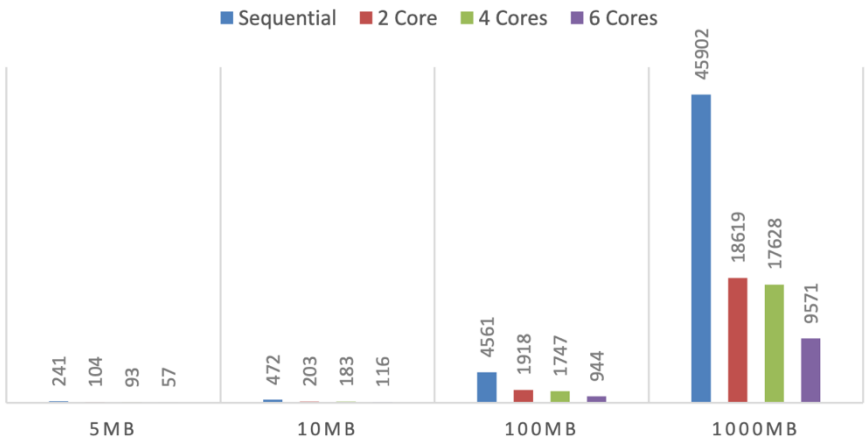


Рисунок 4.16 – Час виконання алгоритму Serpent

Показує отримані значення прискорення послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму Serpent показано в таблиці 4.17.

Таблиця 4.17 – Прискорення алгоритму Serpent

Serpent Speedup	**5MB**	**10MB**	**100MB**	**1000MB**
Sequential	1.00	1.00	1.00	1.00
2 Core	2.32	2.33	2.38	2.47
4 Cores	2.59	2.58	2.61	2.60
6 Cores	4.23	4.07	4.83	4.80

Відмінності у значеннях прискорення послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму Serpent зображено на рисунку 4.17.

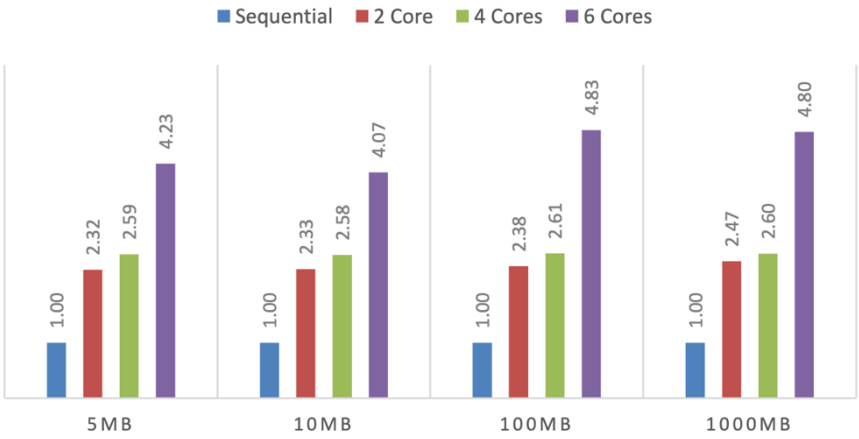


Рисунок 4.17 – Значеннях прискорення алгоритму Serpent

Отримані значення ефективності послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму Serpent показано в таблиці 4.18.

Таблиця 4.18 – Ефективність алгоритму Serpent

Serpent Efficiency	**5MB**	**10MB**	**100MB**	**1000MB**
Sequential	1.00	1.00	1.00	1.00
2 Core	1.16	1.16	1.19	1.23
4 Cores	0.65	0.64	0.65	0.65
6 Cores	0.70	0.68	0.81	0.80

Відмінності у значеннях ефективності послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів для алгоритму Serpent зображено на рисунку 4.18.

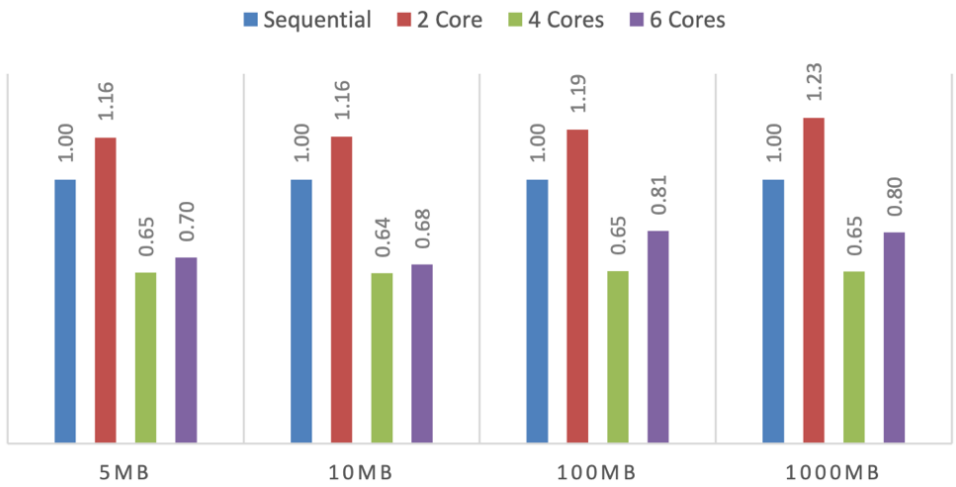


Рисунок 4.18 – Значення ефективності алгоритму Serpent.

Результати RSA алгоритму представлені в таблиці 4.19. На першому тесті я розробив програму, що працює в традиційному режимі (використовучи лише CPU для обчислень). А на другому випробуванні, я використав структуру CUDA для запуску алгоритму RSA в багатопотоковому режимі. У першому стовпці відображається кількість вхідних даних до алгоритма, елементи таблиці - час виконання у мілісекундах.

Таблиця 4.19 – Результати досліджень RSA алгоритму

Кількість вхідних даних	CPU	10 Потоків	100 Потоків	500 Потоків
5000	8281.54	13054.59	1324.57	262.54
10000	16640.61	27211.43	2812.45	524.78
15000	25437.43	41547.81	4189.36	783.47
20000	33485.69	52998.38	5276.58	1047.85
25000	41575.35	66147.85	6708.61	1310.18
30000	49238.84	79328.97	8152.74	1586.64
35000	57619.77	92244.76	9196.49	1854.72
40000	65782.59	105589.37	11280.92	2134.56

У таблиці 4.19 показано залежність між кількістю даних (кількість символів типу char) введення в алгоритм RSA та часом виконання (в мільсекундах) в режимі без потоків та режимі з кількома потоками.

Підвищення продуктивності виконання за допомогою CUDA можна візуально продемонструвати на рисунку 4.19.

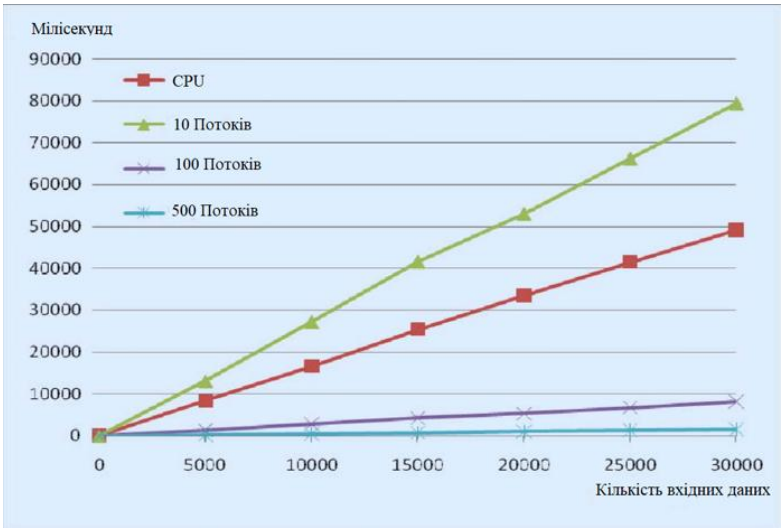


Рисунок 4.19 – Порівняння часу виконання чотирьох режимів

З рисунку 4.19 видно взаємозв'язок між часом виконання та кількість вхідних даних (кількість символів) лінійна для певного режиму. Коли ми використовуємо 500 ниток для виконання алгоритму RSA час виконання дуже короткий порівняно з традиційним режимом. Тож ми можемо сказати, коли

число ниток збільшується, час роботи буде значно зменшено. Крім того, коли число символів збільшується з 5000 до 30000, час виконання 500 ниток майже не збільшуються, що лише доводить погляд, що чим більше кількість потоків, тим коротший час виконання [22].

На рисунку 4.20 показано кількість елементів в мілісекунду в порівняння з традиційним режимом і режимом з кількома нитками за умови 5000 символів введення.

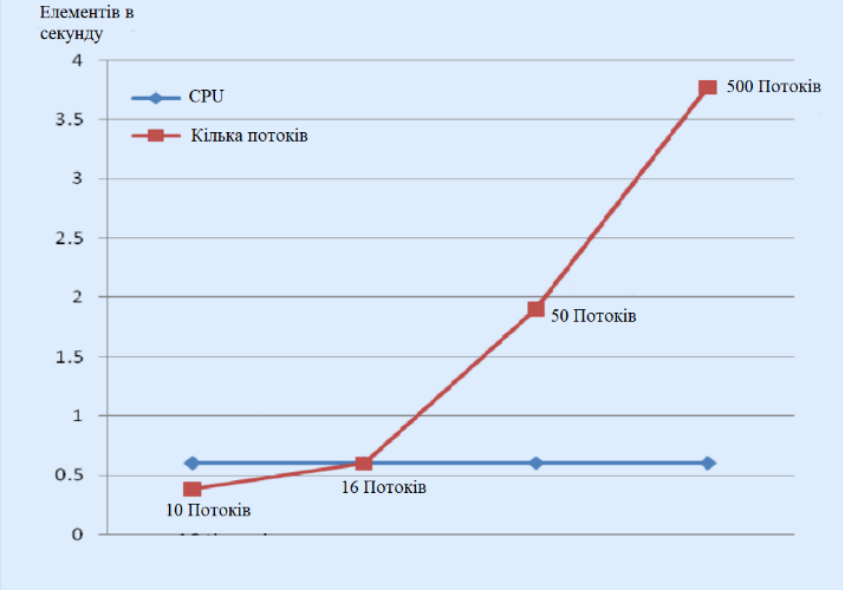


Рисунок 4.20 – Порівняння між традиційним режимом і багатопотоковим режимом

Зауважимо, що коли кількість потоків не велика, наприклад, 10 потоків, час виконання навіть довший, ніж у традиційному режимі. Це пов'язано із співпрацею між процесором та графічним процесором, яка також потребує певного часу, наприклад скопіювати дані з хоста на пристрій. Це також тому, що один обчислювальний блок GPU недостатньо потужний, GPU може обробляти складні обчислювальні задачі ефективно лише тоді, коли є багато потоків, що працюють паралельно. І із зростанням кількості потоків, час виконання буде скорочуватися. Коли кількість потоків доходить до 500, час виконання значно скорочується. Результати експерименту показують, що швидкість руху лінійна, Sp = P, P - кількість потоків в GPU.

Коли звичайний чи зашифрований текст настільки великий, що він не підходить для обробки в одному вузлі ми можемо використовувати HDFS(Handoop Distributed Filesystem), надані компанією Apache технології

Hadoop для розподілу тексту до вузлів кластеру, і кожен вузол кластеру буде викликати ядра CUDA через JCUDA.

4.3 Висновки до розділу

У даному розділі було проведено аналіз криптографічних алгоритмів по результатах дослідження. Дані були проаналізовані на основі послідовної та паралельної реалізації різними процесорними ядрами з чотирма різними розмірами файлів і потім оцінені по трьом критеріям: часу виконання алгоритму, прискоренню та ефективності.

5 ОПИС ПРОГРАМНОГО ТА ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

5.1 Засоби розробки

Розробка проекту проводиться за допомогою мови програмування C++. Дана мова буде зручн у реалізації алгоритмів. Інтерфейс та взаємодія користувачів забезпечується через консольний термінал. Реалізація проекту використовує архітектуру CUDA та PPL (Parallel Patterns Library). Далі буде розглянуто переваги використаних технологій та мов програмування.

C++. Швидкість роботи програм на C++ практично не поступається програмам на C, хоча програмісти отримали в свої руки нові можливості і нові засоби. На мові C++ розробляють програми для найрізноманітніших платформ і систем. Можливість роботи на низькому рівні з пам'яттю, адресами, портами (що, при необережному використанні, може легко перетворитися на недолік). На цій мові є можливість створення узагальнених алгоритмів для різних типів даних, їхня спеціалізація, і обчислення на етапі компіляції, з використанням шаблонів. Підтримуються різні стилі та технології програмування, включаючи традиційне директивне програмування, ООП, узагальнене програмування, метапрограмування (шаблони, макроси).

Бібліотека паралельних шаблонів (PPL) забезпечує імперативну модель програмування, яка сприяє масштабованості та простоті у використанні для розробки одночасних додатків. PPL ґрунтується на компонентах планування та управління ресурсами в режимі одночасного виконання. Це підвищує рівень абстракції між вашим кодом програми та базовим механізмом розпаралелювання, забезпечуючи загальні, безпечні для алгоритмів контейнери, які паралельно діють на дані. PPL також дозволяє розробляти додатки такого масштабу, надаючи альтернативи спільному стану.

PPL надає такі функції:

– паралелізм завдань: механізм, що працює над Windows ThreadPool для виконання декількох робочих елементів (завдань) паралельно;

– паралельні алгоритми: загальні алгоритми, які працюють над версією часу одночасності, щоб паралельно діяти на збір даних;

– паралельні контейнери та об'єкти: загальні типи контейнерів, які забезпечують безпечний одночасний доступ до їх елементів.

CUDA - це модель паралельної обчислювальної платформи та інтерфейсу прикладного програмування (API), розроблена компанією Nvidia. Це дозволяє розробникам програмного забезпечення використовувати графічний процесор (GPU) з підтримкою CUDA для обробки загального призначення - підхід, який називається GPGPU (загальнооб'єдноване обчислення на блоках графічної обробки). Платформа CUDA - це програмний рівень, який забезпечує прямий доступ до набору віртуальних наборів графічних процесорів та паралельних обчислювальних елементів для виконання обчислювальних ядер. Платформа CUDA призначена для роботи з мовами програмування, такими як C та C ++. Ця доступність спрощує паралельне програмування фахівцями використовувати ресурси GPU, на відміну від попередніх API інтерфейсів, таких як OpenGL, які вимагають високих навичок графічного програмування.

5.2 Вимоги до технічного забезпечення

5.2.1 Загальні вимоги

Представлений програмний продукт являє собою программу, для роботи якої, необхідна робоча странція з наступним технічним забезпеченням:

а) технічне забезпечення:

1) процесор з тактовою частотою не нижче 2 ГГц;

2) об'єм оперативної пам’яті не менше 2 Гб;

3) ПЗУ типу SSD або HDD об’ємом не менше 20 ГБ;

б) програмне забезпечення:

1) операційна система – Windows версії 7 і вище;

2) інтерпретатор C++;

3) PPL бібліотека для C++;

4) Nvidia відеокарта;

5.3 Архітектура програмного забезпечення

5.3.1 Опис функціональної моделі

Для проектування діаграми використання спочатку необхідно визначити дійових осіб (акторів), а потім визначити, які дії у системі може виконувати кожен з акторів. В даній системі є два типи дійових осіб – користувачі та система.

Система. Основними функціями системи буде розрахунок та аналіз поданної користувачем інформації.

Користувач. Користувач має змогу вибрати данні для аналізу та обрахунку та визначити типи вихідної інформації. Тепер визначимо дії, які можуть виконувати актори системи (рисунок 5.1).

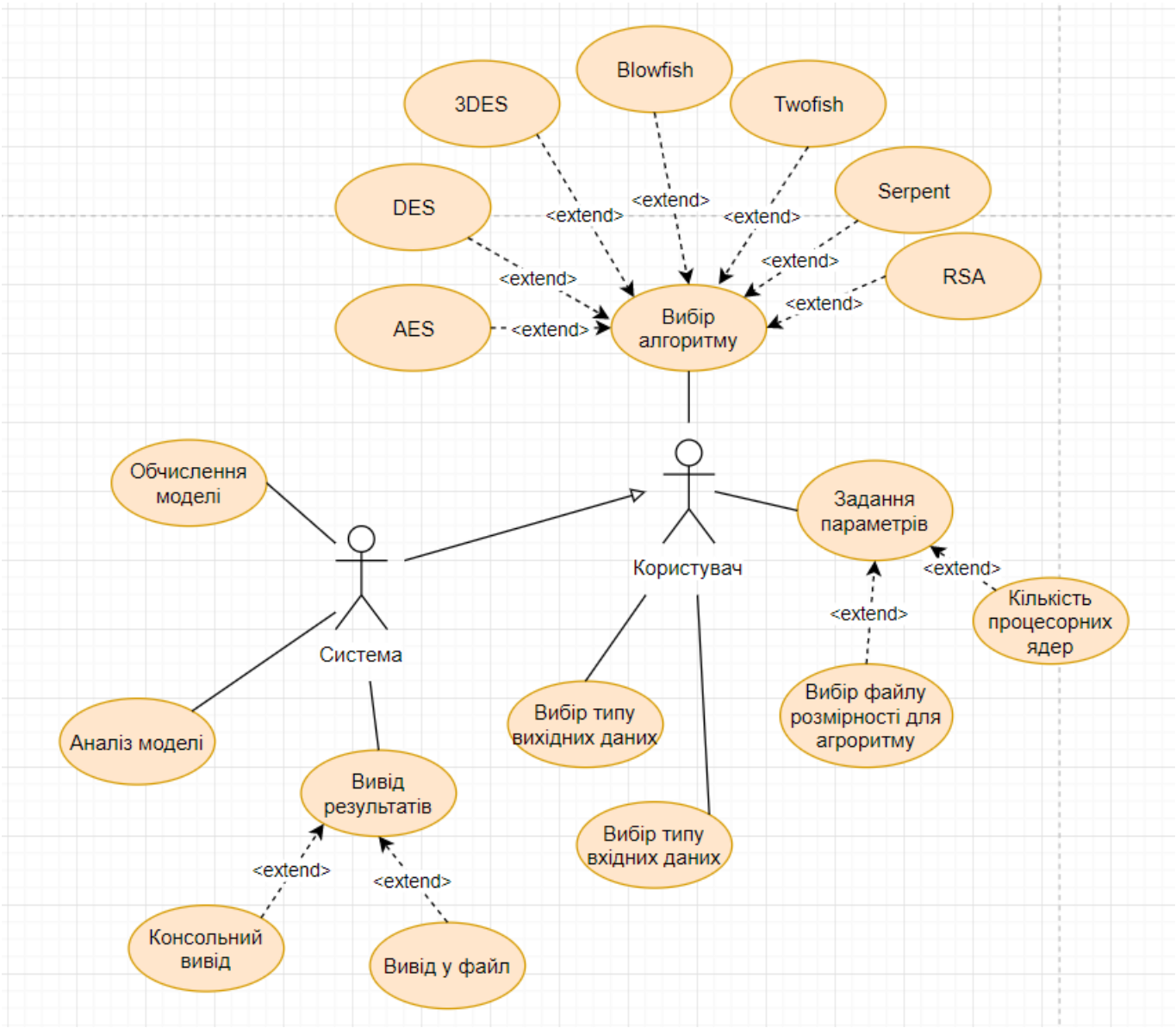


Рисунок 5.1 – Схема структурна варіантів використання. Робота користувача з системою

Список вимог на основі діаграми варіантів використання з їх пріоритетами наведено у таблиці 5.1.

Таблиця 5.1 - Функціональні вимоги

Актор	**Варіант використання**	**Функціональна вимога**	**Пріорітет**
Користувач	Вибір типу вхідних даних	1. Користувач переходить до пункту обрання даних у головному меню. 1.1 Користувач обирає “обрати тип вихідних даних” 1.3. Система відображає доступні типи 1.4 Користувач обирає необхідний тип даних	бажаний
Користувач	Вибір типу вхідних даних	2. Користувач переходить до пункту обрання даних у головному меню. 2.1. Користувач обирає “обрати тип вхідних даних” 2.2. Система відображає доступні типи 2.3. Користувач обирає	бажаний
Користувач	Вибір методу розв’язання	3. Користувач переходить до пункту обирання алгоритму розв’язання у головному меню. 3.1. Система відображає список доступних алгоритмів розв’язання	обов'язковий

Продовження таблиці 5.1.

Користувач	RSA алгоритм	4. Користувач переходить до пункту обирання алгоритму розв’язання у головному меню. 4.1. Система відображає список доступних алгоритмів розв’язання 4.2 Користувач обирає алгоритм RSA	бажаний
Користувач	DES алгоритм	5. Користувач переходить до пункту обирання алгоритму розв’язання у головному меню. 5.1. Система відображає список доступних алгоритмів розв’язання 5.2 Користувач обирає	бажаний
Користувач	3DES алгоритм	6. Користувач переходить до пункту обирання алгоритму розв’язання у головному меню. 6.1. Система відображає список доступних алгоритмів розв’язання 6.2 Користувач обирає алгоритм 3DES	бажаний

Продовження таблиці 5.1.

Користувач	AES алгоритм	7. Користувач переходить до пункту обирання алгоритму розв’язання у головному меню. 7.1. Система відображає список доступних алгоритмів розв’язання 7.2 Користувач обирає Алгоритм AES	бажаний
Користувач	Blowfish алгоритм	8. Користувач переходить до пункту обирання алгоритму розв’язання у головному меню. 8.1. Система відображає список доступних алгоритмів розв’язання 8.2 Користувач обирає	бажаний
Користувач	Twofish алгоритм	9. Користувач переходить до пункту обирання алгоритму розв’язання у головному меню. 9.1. Система відображає список доступних алгоритмів розв’язання 9.2 Користувач обирає Алгоритм Twofish	бажаний

Продовження таблиці 5.1.

Користувач	Serpent алгоритм	10. Користувач переходить до пункту обирання алгоритму розв’язання у головному меню. 10.1. Система відображає список доступних алгоритмів розв’язання 10.2 Користувач обирає Алгоритм Serpent	бажаний
Користувач	Кількість процесорних ядер	11. Користувач переходить до пункту обирання початкових даних у головному меню. 11.1. Система відображає список пунктів підменю “задання початкових даних” 11.2. Користувач обирає пункт “задання параметрів” 11.3. Система відображає список пунктів підменю “ задання параметрів ” 11.4. Користувач обирає пункт “кількість процесорних ядер” у	бажаний
Користувач	Задання параметрів	12. Користувач переходить до пункту обирання початкових даних у головному меню. 12.1. Система відображає список пунктів підменю “задання початкових даних” 12.2. Користувач обирає пункт “задання параметрів” 12.3. Система відображає список пунктів підменю “	бажаний

Продовження таблиці 5.1.

Система	Вивід результатів	13. Система відображає результати в форматі обраним користувачем	обов'язковий
Система	Вивід у файл	14. Система відображає результати в форматі обраним користувачем 14.1. Вихідна інформація виводиться у файл	обов'язковий
Система	Консольний вивід	15. Система відображає результати в форматі обраним користувачем 15.1. Вихідна інформація виводиться в консоль	обов'язковий
Система	Аналіз моделі	16. Система аналізує структуру моделі	обов'язковий
Система	Обчислення моделі	17. Система обчислює модель обрану користувачем за обраним	обов'язковий
Система	Розбиття моделі	18. Система обчислює модель обрану користувачем за обраним користувачем алгоритмом 18.1. Система обробляє модель для обчислення	обов'язковий

5.3.2 Опис процесу діяльності

Розглянемо дії які періодично буде виконувати система при виборі користувачем введення даних, за допомогою UML діаграми діяльності (рисунок 5.2).

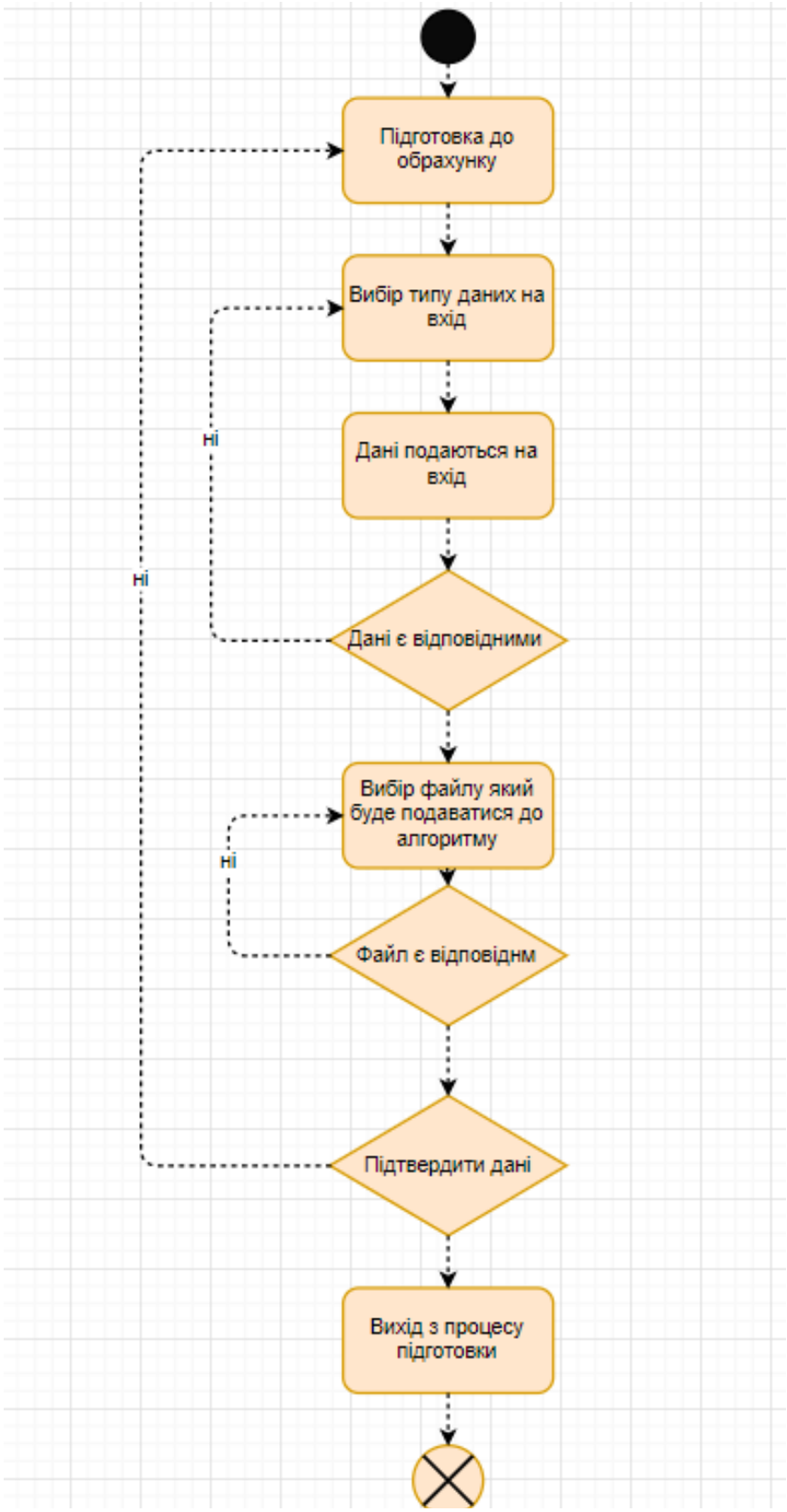


Рисунок 5.2 – Схема структурна діяльності. Процес введення даних для обробки

Детальний опис схеми структурної діяльності процесу внесення початкових даних наведено у таблиці 5.2.

Таблиця 5.2 – опис діяльності процесу внесення даних

Варіант діяльності	**Опис варіанту діяльності**
Підготовка до обрахунку	Загальна підготовка до введення даних (Головне меню)
Вибір типу даних на вхід	Обирається тип даних, який буде подаватися на вхід системи
Дані подаються на вхід	Дані подаються на вхід до системи
Дані є відповідними	Перевірка на співпадіння даних які поданих на вхід із зарані обраним типом. Якщо, тип не співпадає – то повернення до вибору типу вхідних даних.
Вибір файла алгоритму	Файл алгоритму подається на вхід до системи
Файл дійсний	Перевірка на дійсність файлу поданного на вхід. Якщо, обраний файл не є дійсним – то повернення до вибору файлу алгоритму
Вибір методу розв’язання	Обирається метод, який буде використаний при подальшому розрахунку
Потрібні додаткові параметри	Обрання необхідності додання додаткових параметрів до даних наданих системі. Якщо, відповідь затверджена – то перехід до введення додаткових параметрів.
Введення додаткових параметрів	Додаткові параметри подаються на вхід до системи
Вибір типу вихідних даних	Обирається тип даних, який буде отриманий на виході із системи
Підтвердити дані	Перевірка на дійсність та коректність усієї поданої інформації. Якщо, інформація та дані незадовільні то перехід до підготовки до
Вихід з процесу підготовки до розрахунку	Перехід до процесу аналізу та роботи с обраними даними.

5.4 Висновки до розділу

Розглянуто основні особливості розробки програмних застосувань із криптографічними алгоритмами на мові програмування C++ з використанням бібліотеки PPL. До системи, що розробляється були поставлені мінімальні технічні вимоги для функціонування системи. Розроблена схема архітектури програмного забезпечення повністю відповідає поставленим вимогам. Під час розробки було сформовано та розроблено ієрархію класів, які відповідають поставленим вимогам та забезпечують необхідний функціонал системи. Описані особливості розробки системи, архітектура програмного забезпечення та елементи які взаємодіють між собою в системі.

6 РОЗРОБКА СТАРТАП ПРОЕКТУ

Стартап: на технології блокчейн розробка продукту для медичної та адміністративної галузей, який буде запобігати підробці документів, а саме довідок, лікарняних листів тощо.

Таблиця 6.1 – Опис стартап проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Розробка продукту (веб-платформи) на технології блокчейну, який буде запобігати підробці документів, а саме довідок, лікарняних листів тощо.	1. Медична галузь 2. Адміністративна галузь	1. Можливість швидко та легко отримати будь-яку довідку 2. Цифрова форма довідки, яку легко зберігати. 3. Неможливість підробки документів. 4. Заощадження на безпаперовій роботі держави 5. Полегшення збору податків

Світ блокчейна дуже сильно фрагментований - і велика кількість можливостей імплементації, технічна складність теми, а також медійний ажіотаж навколо індустрії дуже заважають відокремити найголовніше.

Блокчейн – одна з найбільш трендових технологій сьогодення (Рисунок 1), яка дає змогу відкрити дані й продемонструвати, що вони не зазнали змін, не розкриваючи при цьому персональні дані користувачів. Через прозорість і незалежність від урядів та корпорацій його називають проривом XXI ст. [23]. В середині січня 2016 р. аналітична компанія CB Insights опублікувала список найбільших галузей, де може застосовуватися блокчейн [24] – це фінанси, логістика, менеджмент, медицина та ін.

Естонія, яка є лідером по впровадженню електронних державних послуг, спільно з блокчейн-стартапом Guardtime впроваджує створення єдиної бази медичних книжок для населення, які будуть доступні для обміну інформацією клінікам, а також страховим компаніям. Теж саме робить компанія Prescrypt

(спільно з SNS Bank і Deloitte) в Нідерландах, і BitHealth в США. Шведські чиновники спільно з ChromaWay і банком-партнером створюють єдиний реєстр земельних ділянок на блокчейні - щоб полегшити життя продавцям і покупцям, а також банкам, які хочуть використовувати їх як заставу. Такий же пілот в Грузії робить BitFury, BitLand в Гані (збираються масштабуватися в Нігерію і Кенії), і Гондурас. ОАЕ збирається перевести весь державний документообіг на блокчейн до 2020 року. Штат Делавер, в якому зареєстровано безліч компаній з інших штатів і держав, вводить систему реєстрації компаній, випуску акцій, фіксування рішень рад директорів, перерозподілу часток в результаті купівлі-продажу на блокчейні (такий же функціонал реалізує в декількох державах сінгапурська компанія Otonomos). Британська компанія Everledger надає трекінг і провенант для діамантів, предметів мистецтва та дорогого алкоголю.

За опитуванням учасників одного з останнього Всесвітнього економічного форуму вже до 2023 року технологія блокчейну буде активно використовуватися в сфері державних послуг провідними світовими державами. Більш того, близько 10% світового ВВП (за прогнозами ОЕСР) буде створюватися при безпосередньому використанні технології блокчейну. Основні переваги від впровадження технології очікуються в скороченні операційних витрат (73% опитаних), скорочення часу розрахунків (69% опитаних), скорочення ризиків (57% опитаних), зростанні можливості отримання додаткових доходів (51% опитаних). В Україні ця технологія дозволить кожному з нас, як на машині часу, ривком, а по-іншому ми не вміємо, перейти від неефективної бюрократичної державної системи співжиття з рідною державою, до сучасної, необтяжливою, зручною, чесної системи «держава - це я» .

Держсектор є складним і інертним механізмом, залишаючись при цьому централізованою системою. Від розвиненості цієї системи залежить ефективність держуправління як такого, рівномірне покриття державними послугами потреб населення і підприємців (наприклад, реєстрація компанії, шлюбу, отримання довідок та виписок). Часом на складнощі у взаємодії людини і держапарату, його непрозорості виростають цілі індустрії посередників (допомога в реєстрації ТОВ, заповнення довідок ДАІ і т.д.). Чим більше посередників - тим дорожче і складніше послуга. Організаційні структури держапарату часто фрагментовані і майже завжди

розрізнені, що робить складним обмін інформацією між департаментами і відомствами. Часто посередники в ланцюжку отримання держпослуг невидимі одержувачу.

Багато країн усвідомлюють запити нового покоління, людей, які звикли до швидких і зручних продуктів, проводять дослідження і вирішують вищезазначені проблеми - активно реформують систему надання держпослуг. Деякі мобілізують розділені IT департаменти в єдині системи - так звані «agencies»; інші починають використовувати альтернативні глибокі дані і «dark analytics» для швидкого аналізу кореспонденції і запитів з боку населення; треті розробляють нові архітектури взаємодії між державними юнітами; ну а останні, найбільш просунуті, звичайно ж, застосовують технології розподіленого реєстру (блокчейн), на чому ми і зупинимося.

Швидкість, з якою різні державні департаменти країн світу сьогодні починають цікавитися застосуванням блокчейн-технологій не пропорційна реальному розвитку і рівню практичного впровадження даної технології - що цілком логічно і зрозуміло. Система держуправління повинна бути стійкою, це вкрай статичний, малорухливий механізм, і будь-який впровадження повинно довести свою ефективність. До того ж не всі блокчейн-рішення здатні масштабуватися і відповідати реальному навантаженню. Але незважаючи на це, є ряд дійсно цікавих проектів, про які можна розповісти.

З практичної точки зору, на мій погляд, Естонія не є світовим інноваційним бенчмарком, але, якщо поглянути на їх державний проект єдиної державної електронної системи - це один з найуспішніших з реалізованих в світі. Проект став успішним завдяки особливій інфраструктурі: замість єдиної центральної системи, була створена децентралізована відкрита система, яка з'єднує між собою різні сервіси та бази даних. Завдяки такій структурі системи, вбудовування в неї нових сервісів і додатків стало вкрай простим, а переклад їх на основу блокчейну проходить з меншими витратами і суперечками, ніж при централізованій роботі держави.

Ключові результати всієї системи були наступними: в 2016: 94% громадян мають електронне посвідчення, що дозволяє користуватися системою; 2% ВВП

країни зекономлено на безпаперовій роботі держави; 4000+ сервісів надаються електронно; Естонія - країна №1 в світі по збору податків і за Індексом Електронної Економіки.

Охорона здоров'я. Незважаючи на те, що електронні медичні картки, онлайн-доступ до даних пацієнта і їх зміна можуть бути реалізовані без використання блокчейн, проблема достовірності та надійності даних залишається невирішеною. При використанні блокчейн-технології несанкціонованих змін / доступ / використання даних громадян стає неможливим, так як будь-яка інформація про подібні дії записується в системі.

У Голландії в 2016 році компанія Prescrypt у співпраці з SNS Bank NV і Deloitte розробила блокчейн-додаток, що робить більш легкими і доступними послуги для хронічно хворих пацієнтів. Концепція використовує Idin-сервіс онлайн-аутентифікації, що надається банками, як засіб для підключення до блокчейну. Idin забезпечує таку саму безпеку і зручність, як інтернет-банкінг.

Технологія блокчейна може допомогти в боротьбі з корупцією. У деяких країнах її вже почали використовувати в тестовому режимі на державному рівні.

У країнах, де кожен може змінити публічні записи в держреєстрі, якщо заплатить, кому треба, корупція є гострою проблемою. Її рішенням може стати технологія блокчейна, яка, до речі, вже успішно застосовується в Грузії. В її блокчейн-системі зберігається майже 200 тисяч записів про права на земельні ділянки, причому з одного боку вона надійно захищена від несанкціонованого доступу, а з іншого - будь-який бажаючий може переглянути всі ці записи. Це перший випадок використання децентралізованої мережі на державній службі.

Проект допомагав розробляти акселератор Blockchain Trust Accelerator. Його співзасновник Томик Тіллеман раніше працював в Державному департаменті США і написав промову Хілларі Клінтон про свободу інтернету (для цього йому довелося пропрацювати 100 годин без сну).

Тіллеман вірить, що блокчейн допоможе зберегти цілісність загальнодоступних данних, будь то інформація про власників землі або кількість голосів на виборах. «Зараз суспільство насилу довіряє уряду, - сказав він. - Технологія блокчейна зможе

вирішити цю проблему, адже вона надає надійні і відкриті системи, які не зможе торкнутися корупція ».

За даними соціологічного опитування компанії Edelman, в 2017 році довіра людей до державних інституцій знизилася. Жителі 14 країн, які брали участь в опитуванні, зізналися, що найменше довіряють своєму уряду.

У Грузії виникла потреба в такій системі ще в хаотичні пострадянські часи, коли чиновників часто звинувачували в маніпуляціях із записами в держреєстрі. Зараз же уряду країни вдалося очистити свою репутацію, і Тіллеман вважає, що це тільки початок. Проект кадастрової карти на блокчейні був пілотним, слідом за ним уряд Грузії планує перевести на цю технологію і інші реєстри та сервіси. Відкритість і прозорість блокчейна допоможе повернути довіру до державної влади.

Поява і розвиток блокчейн-технології змусило задуматися про принципово нові можливості реалізації e-Government. За результатами звіту «Зміцнення довіри до уряду» (2017, січень), підготовленого IBM Institute for Business Value (IBV), «дев'ять з десяти керівників країн планують в 2018 році інвестувати в розробку блокчейн- рішень в області фінансових операцій, управління активами, управління контрактами і дотримання нормативних вимог».

Перше очевидне і найбільш важливе для e-Government перевага блокчейна в порівнянні зі стандартними базами даних - це захищеність інформації від фальсифікації. Це означає, що дані про громадян, нерухомость, компанії, сертифікаті, дипломі, права на власність та ін. Після занесення в державні блокчейн-реєстри змінити практично неможливо. Найголовнішим наслідком такої граничної надійності є можливість використовувати дані реєстра в якості повноцінних юридичніх документів: запис у блокчейн-реєстрі стає більш достовірним будь-якого паперу з підписом і печаткою та до того ж доступна завжди і всюди.

Другим найважливішим перевагою блокчейн-платформ для побудови e-Government є можливість використання механізму смарт-контрактів для автоматизації операцій з даними. Якщо блокчейн-реєстри містять юридично правомочні записи, скажімо, про власність, то механізм передачі цієї власності, а по суті, процедуру внесення до реєстру запису про нового власника, можна доручити спеціальній програмі - смарт-контракту. І якщо контракт зберегти в блокчейні,

таким чином виключивши можливість його несанкціонованої зміни, і одночасно забезпечити однозначність виконання алгоритму контракту (в будь-який момент часу, на будь-якому вузлі мережі блокчейн), то йому, як і записів в реєстрах, можна привласнити юридичну значимість. При цьому слід звернути увагу на те, що регламентують документи - державні закони та інші нормативні акти - здебільшого описують алгоритми дій з даними реєстрів. Тому є можливість формулювати їх на програмній мові смарт-контрактів і також помістити в блокчейн, де вони, набувши статусу смарт-законів, будуть виконуватися автоматично.

Виходить, що використання блокчейна може перевести саму ідею електронного уряду на новий рівень. Слід вести мову вже не просто про зручний сервісі надання громадянам і бізнесу, а про принципову переформатування самої діяльності держави, повне занурення її в цифрову екосистему блокчейна.

В результаті держава отримає значне скорочення бюрократичного апарату, практичне усунення паперового документообігу, істотне зниження транзакційних витрат, повний контроль над діяльністю чиновників, ну і найголовніше, створення сприятливого середовища для бізнесу і громадян.

На хвилі інтересу урядів як розвинених, так і бідних країн до блокчейн-технології з'явилися і стартапи, вже з самого початку орієнтовані на побудову блокчейн-платформ спеціально для E-government: австрійська Neocapita, Procivis в Швейцарії та Естонії.

Neocapita заявила про створення децентралізованої платформи на основі private fully-permissioned blockchain Stoneblock, призначеної для вирішення найдорожчою проблеми електронного уряду - створення реєстрів. Технічний опис відсутній. Однак 29 березня в Facebook був анонсований випуск Stoneblock Whitepaper. Так само інформація, що Neocapita веде переговори про впровадження платформи Stoneblock в Афганістані і в Папуа-Новій Гвінеї.

Швейцарський стартап Procivis у співпраці з експертами електронного уряду з Естонії анонсували запуск до кінця 2017 пілотної версії «app store» для електронного уряду на базі блокчейна. Procivis ставить завдання створити додатки, що реалізують повний спектр послуг для громадян - таких, як цифрова ідентифікація, голосування, подача податкових декларацій, ведення кадастру та ін.

6.1 Сфери поширення технології блокчейн в Україні

Державний аукціон з продажу конфіскованого майна «Сетам». У вересні 2017 року в Україні презентували систему державних аукціонів з продажу конфіскованого майна "Сетам", побудовану на основі технології блокчейн. А в жовтні того ж року запрацював Державний земельний кадастр України теж з використанням блокчейну. На думку експертів, щоб якісно змінити відносини між державою та її громадянами, треба також перевести на цю технологію й решту баз даних: від Держреєстру речових прав на нерухоме майно до Державного реєстру виборців.

Електронні аукціони на блокчейн. Ще у березні 2016-го в Києві був підписаний Меморандум про розвиток і впровадження системи децентралізованих онлайн-аукціонів в державних галузях. Платформу Eauctoin 3.0 створювали (IDF Reforms Lab, Distributed Lab, Ощадбанк, Приватбанк, Microsoft та Unitybars). На початку липня 2016 року відбувся перший блокчейн-аукціон, якій дозволив здавати в оренду державне майно.

Відкрита платформа електронної демократії. E-vox – ініціатива, спрямована на підвищення прозорості в державному управлінні шляхом використання блокчейн в організації голосувань, референдумів, підписанні петицій тощо. У серпні 2016 року команда E-vox встановила систему голосування в мерії міста Балта (Одеська область) – депутати змогли голосувати прямо зі своїх смартфонів і планшетів, залишаючи запис в блокчейн. Згодом системою будуть користуватися більше державних установ.

Блокчейн в НБУ. На початку листопада 2016 року під час конференції Cashless Ukraine Summit в Києві НБУ представив дорожню карту розвитку безготівкової економіки. Проект передбачає створення альтернативи картковим розрахункам. З 2019 року може розпочатись випуск електронних грошей на базі блокчейн. Революція в роботі з електронними грошима відбуватиметься в рамках розвитку національної платіжної системи “ПРОСТІР”.

Банки на блокчейн. Українські банки також починають працювати з електронними грошима на блокчейн. За допомогою розподіленої банківської системи Smart Money банки будують інфраструктуру для операцій з електронними грошима.

Електронний уряд. У березні 2016 року на Blockchain Conference Kiev презентували концепцію порталу електронного уряду – E-Ukraine. Тоді платформа була ще на стадії написання технічного забезпечення і мала стати точкою взаємодії громадян, бізнесу і держави, що об'єднає на одній платформі аукціони, голосування, ведення держреєстрів і інші розподілені сервіси.

Електронна гривня. У НБУ розглядають можливість впровадження електронної національної валюти – е-гривні. Основою нової валюти стане технологія блокчейн. Перші кроки з впровадження національної е-валюти на основі технології блокчейн були здійснені ще в кінці 2016 років, коли НБУ спільно з провідними українськими фахівцями в цій галузі розпочав дослідницький проект. Тобто в Україні можуть з'явитися електронні гроші, які не будуть прив'язані до жодних фінансових установ. Електронні гроші є не криптовалютою, а еквівалентом реальних грошей, випуск яких контролює держава. Також електронні гроші є системою моментальних веб-розрахунків. З кінця 2017 року в Україні діє новий криптофонд Vanhealthing Cryptofund of biotech innovations, який розробляє проекти у сфері біотехнології і блокчейн. Його партнерами є декілька організацій, зокрема венчурний фонд USP Capital, компанії Planexta і Sikorsky Challenge. Криптофонд відрізняється від традиційних фондів приватних інвестицій перш за все тим, що збирає кошти через Initial Coin Offering (ICO).

6.2 Висновки до розділу

Система блокчейн максимально спрощує саму процедуру надання адміністративних послуг, зменшує витрати державного і місцевих бюджетів на утримання апарату держави. З іншого боку, у зв'язку з розвитком цієї технології, матимуть місце скорочення і зменшення можливостей для чиновників зловживати доступом до баз даних. Тобто держава «зацікавлена» в гальмуванні цього процесу, не бажаючи втратити контроль за електронними документами. Для блокчейну

неважливо, чи розвинута країна чи ні. Насправді це дуже проста технологія, яка не потребує великих фінансових витрат. Потрібна просто політична воля держави, щоб урядові сервіси щодо різного роду реєстрацій від авто до нерухомості стали доступними і при цьому захищеними.

ВИСНОВКИ

Ця теза стосувалася розгляду різних методик оптимізації, що використовуються в архітектурі x64 при реалізації різних блокових шифрів та застосуванні цих методів при побудові нових більш швидких реалізацій вибраних блокових шифрів. Використання різних методик для конкретного алгоритму блочного шифрування значною мірою залежить від фактичної конструкції алгоритму. Зазвичай блок-шифри розроблені таким чином, що швидкі реалізації програмного забезпечення можуть використовувати великі оглядові таблиці для фаз заміщення та перестановки, знайдених у раудовій функції.

Цей огляд охоплює основні аспекти проекту, які можна розділити на дві важливі частини: безпеку та паралельні частини. Основна мета полягала в описі криптографічних алгоритмів, оскільки вони покладаються на математику і потребують математичного розуміння, щоб мати можливість їх найкращим чином запрограмувати. Особливо це актуально, коли цільовою місією є паралелізація цих алгоритмів, які потребують дуже глибокого розуміння не лише для їх кроків та етапів, але й того, як вони працюють математично. Паралельна частина зосереджена на технологіях, доступних для побудови паралельної програми. Крім того, він зосереджується на заходах щодо ефективності, які можна застосувати для вдосконалення паралельного алгоритму.

Режими роботи, які мають зворотний зв'язок блоку шифротексту, вимагають зашифрувати попередній блок прямого тексту, серіалізуючи процес шифрування. Такі режими роботи не можуть отримати користь від додаткових показників, введених при паралельній обробці. Тому більшість реалізацій блокуються паралельно для досягнення більшої продуктивності.

Планування поза порядком, яке доступне в більшості процесорів x64 сьогодні, може знайти паралелізм рівня інструкцій і паралельно виконувати незалежні потоки інструкцій. Matsui у 2016 році представив цю методику впровадження шифру Camellia у двосторонній паралельний спосіб, переплітаючи дві операції шифрування блоків, щоб ввести більше паралелізму для процесора для використання та отримання більш високої продуктивності. Було успішно застосували цю техніку до

реалізації Blowfish, AES, DES, 3DES та Twofish. Twofish - найкращий вибір серед усіх кандидатів AES через унікальне поєднання швидкості, гнучкості та консервативного дизайну. Якщо припустити, що це безпечно (і покаже лише час), Twofish - найшвидший кандидат AES у всіх процесорах. Twofish підходить для смарт-карт, навіть тих, у яких є лише пара регістрів, кілька байтів оперативної пам'яті.

Жоден інший алгоритм не має такої гнучкості в реалізації: можливість відключення часу налаштування ключа для швидкості шифрування, а також ROM та RAM для швидкості шифрування. Ці параметри існують для 32-бітних процесорів, 8-бітних процесорів та апаратних засобів.

У цих реалізаціях ще є можливість вдосконалитись. Наприклад, лічільник кешів використовувався для реалізації алгоритмів. Цю техніку оптимізації можливо застосувати до інших шифрів. Цю методику було застосовано, щоб отримати порівняльні результати з попередніми дослідженнями.

Було продемостровано роботу RSA алгоритму в архітектурі CUDA. Було проаналізовано паралельний алгоритм RSA, описано методи паралелізації. Вузьке місце для алгоритму RSA полягає в розмірі простого тексту. Як видно з даних дослідження в архітектурі CUDA алгоритм працює з високою продуктивністю, особливо при наявності великої кількості потоків.

Однак є деякі недоліки, які слід вдосконалити. Відомо, що GPU не підтримує багаточну арифметику, таку як обчислити виконання арифметики великих чисел розміром в 1024 біт. GPU підтримує до 64 біт. Отже, якби ми могли використовувати числа з більшим розміром в бібліотеці на GPU, реалізація була б більш ефективною.

ПЕРЕЛІК ПОСИЛАНЬ

1. Задірака В.К., Олексюк О.С. Комп’ютерна арифметика багаторозрядних чисел. //Киі□в – 2003.

2. Карацуба А.А., Офман Ю.П. Умножение многорозрядных чисел на автоматах //ДАН СССР. — 1962. т.145. — С. 293-294.

3. Шенхаге А., Штрассен В. Быстрое умножение больших чисел // Кибернет. сб. — 1973. — вып. 10. — С. 87-98.

4. Cook S. A., Aanderaa S. O. On the minimum computation time of functions, Thesis, Harvard University, 1966. — P. 26-50.

5. Березовский А.И., Задирака В.К., Шевчук Л.Б. О тестировании быстродействия алгоритмов и программ вычисления основных операций ассиметричной криптографии /Кибернетика и системный анализ № 5, 1999. - с. 61-68.

6.Задирака В.К., Игисинов К. Анализ погрешности округления алгоритма быстрого преобразования Фурье и некоторых его приложений для режима с плавающей запятой [Текст] / Задирака В.К. // Киев, 1972 р. (Препр. Ин-т кибернетики им. В.М. Глушкова АН УССР: 72-23). – 593 с.

7.Задирака В.К. О сравнении некоторых алгоритмов вычисления оценок интеграла Фурье в корреляционной функции [Текст] / Математическое обеспечение ЭЦВМ // Киев: Ин-т кибернетики им. В.М. Глушкова АН УССР, 1971 р. – 249-250 с.

8. Чефранов, Олександр Г та Махмуд, Ахмед Й (2013), протокол обміну ключами, заснований на матриці, заснованому на Діффі-Гелмана, інформаційні науки та системи.

9. Daemen, J., & Rijmen, V. (2010). Блок-шифр Rijndae.

10. Daemen, J., & Rijmen, V. (2002). Дизайн Rijndael: AES - вдосконалений стандарт шифрування.

11. Daernen, Joan, Rijrnen, & Vincent. (2002). Дизайн Rijndael, AES - Розширений стандарт шифрування. Берлін: Спрінгер.

12. Fruhwirth, C. (2011). LUKS Специфікація формату на диску.

13. Деніел Пейдж Паралельна криптографічна арифметика з використанням надмірного представлення монтгомерії // Транзакції в комп'ютерах, 2014 р., с. 1474-1482.

14. Шен Лян Посібник для шпигування програміста Java Native Interface (JNI) // 2008 р.

15. JCUDA, http://www.jcuda.de/, 2017 р.

16. (Schneier, Products that Blowfish., 2013a), (Schneier, Products that Use Twofish, 2013b), (Fruhwirth, 2011), (Chan, 2015).

17. Grama, A., Karypis, G., Kumar, V., & Gupta, A. (2003). Вступ до паралельних обчислень.

18.Hasselbring, W., Jodeleit, P., & Kirsch, M. (1998). Реалізація паралельних алгоритмів на основі оцінювання та трансформації прототипів. Обчислювальні опитування ACM.

19. Рогоза А.В. Швидкі алгоритми обчислювання криптопримітивів / А.В. Рогоза // Матеріали Міжнародної наукової інтернет-конференції «Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення». – м.Тернопіль, 11 червня 2019 р. – с.53-54.

20. Рогоза А.В. Ефективні алгоритми обчилення криптопримітивів в паралельній системі обчислень / А.В. Рогоза, В.К. Задірака // Матеріали III всеукраїнської науково-практичної конференції молодих вчених та студентів «Інформаційні системи та технології управління» (ІСТУ-2019) – м. Київ.: НТУУ «КПІ ім. Ігоря Сікорського», 20-22 листопада 2019 р. – с.97-101.

21. Рогоза А.В. Сравнительный анализ криптографических алгоритмов / А.В. Рогоза, В.К. Задирака // материалы LIII Международной научной конференции «Актуальные научные исследования в современном мире» / Москва, г. 2019, с.104-108.

22. Рогоза А.В. Порівняльний аналіз криптографічних алгоритмів / А.В. Рогоза, В.К. Задірака // материали III Международної наукової конференції «Наукові теорії сьогодення та перспективи розвитку наукової думки» (ІСТУ-2019) – м. Київ, р. 2019, с.105-109.

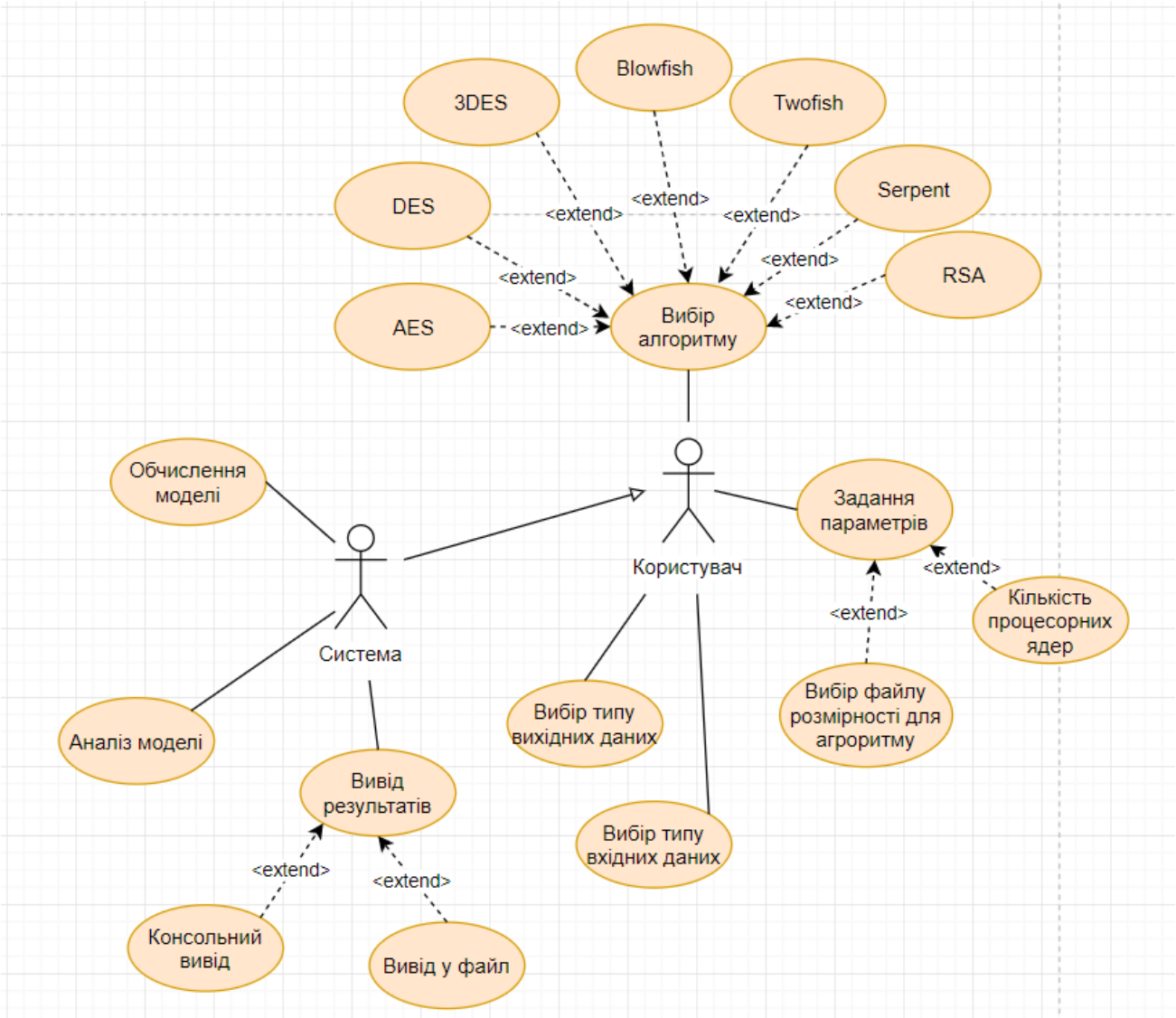
23. Прокопин В.В. Актуальные проблемы стратегического развития экономики, 2015, с. 58-63.

24. Technical information on Bitcoin’s processes, keys and purposes. Available at: https://bitcoin.org/en/hot-it-works (accessed December 1, 2017).

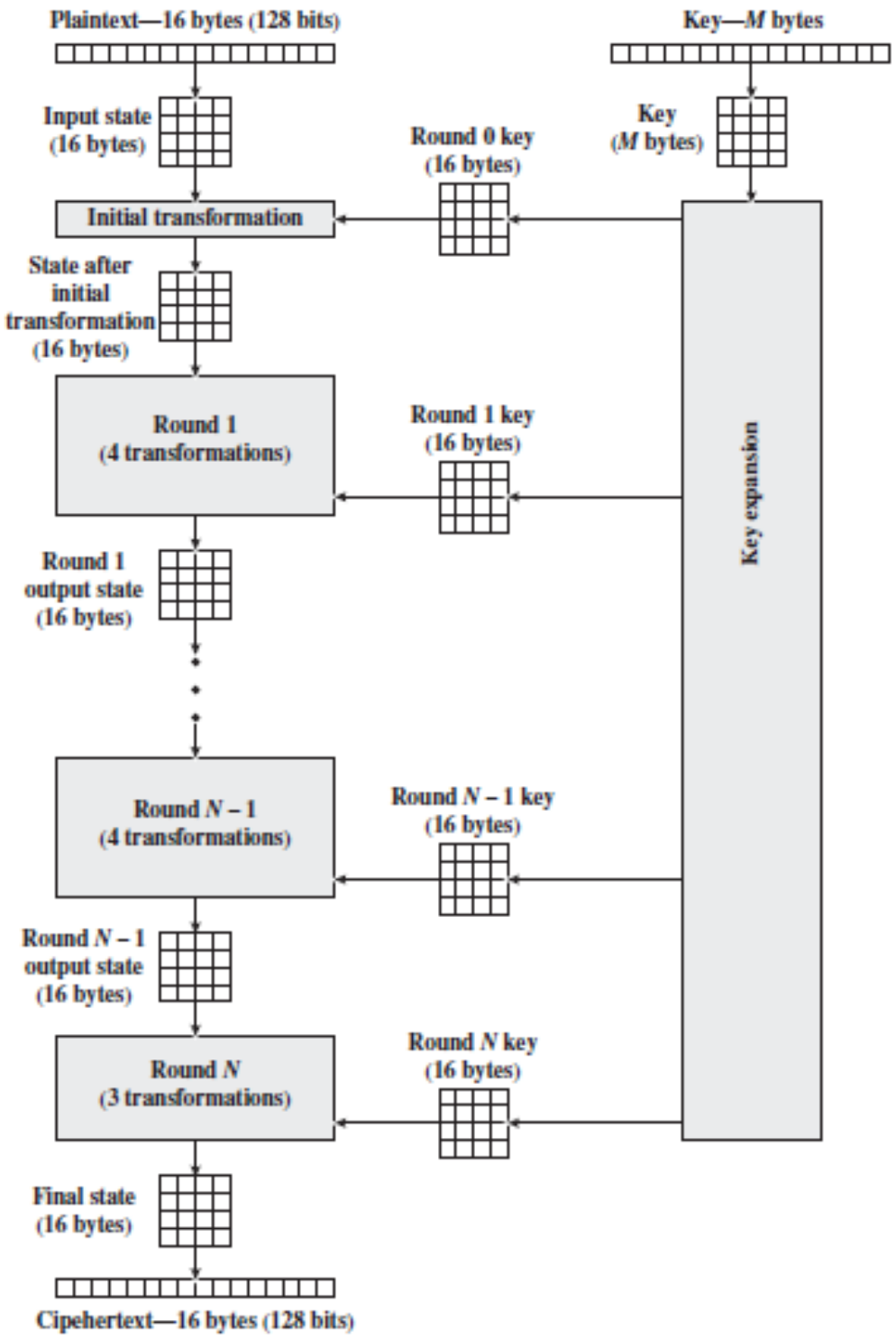
ДОДАТОК А

ГРАФІЧНИЙ МАТЕРІАЛ

ПЛАКАТ 1 СХЕМА СТРУКТУРНА ВАРІАНТІВ ВИКОРИСТАННЯ



ПЛАКАТ 2 СХЕМА РОБОТИ AES АЛГОРИТМУ



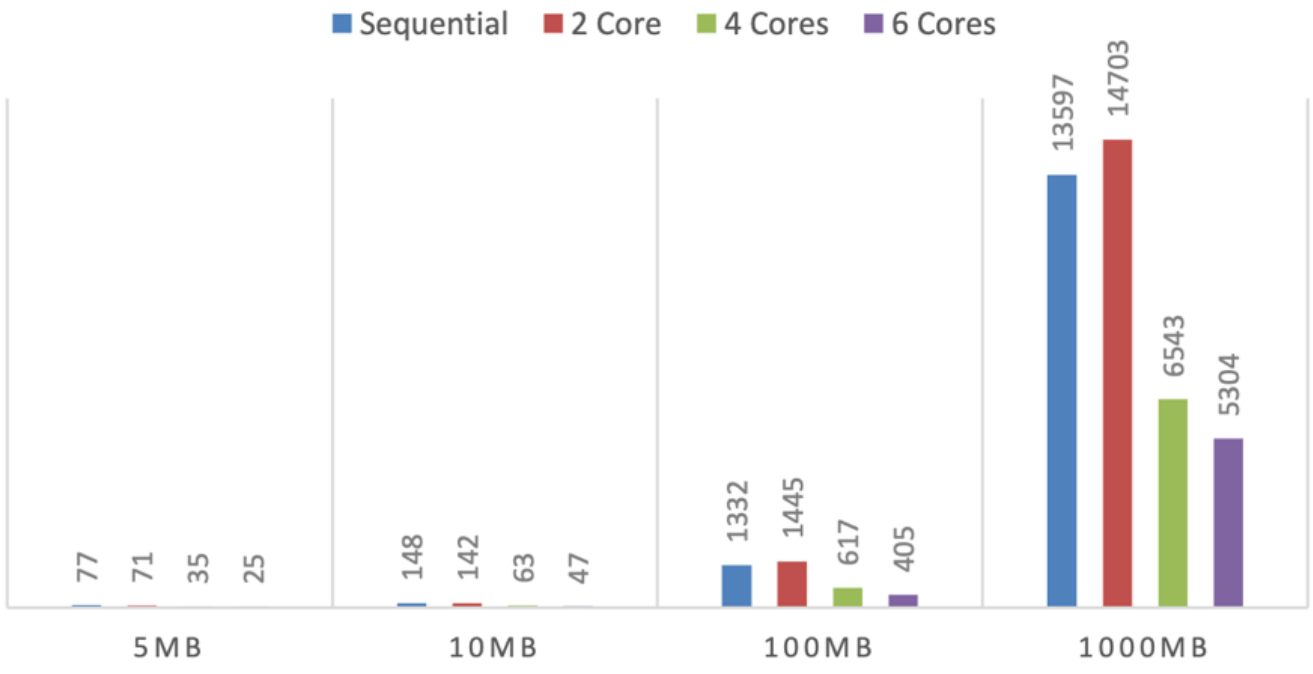
ПЛАКАТ 3 РЕЗУЛЬТАТИ РЕАЛІЗАЦІЇ AES АЛГОРИТМУ

AES Runtime	5MB	10MB	100MB	1000MB
Sequential	77	148	1332	13597
2 Core	71	142	1445	14703
4 Cores	35	63	617	6543
6 Cores	25	47	405	5304

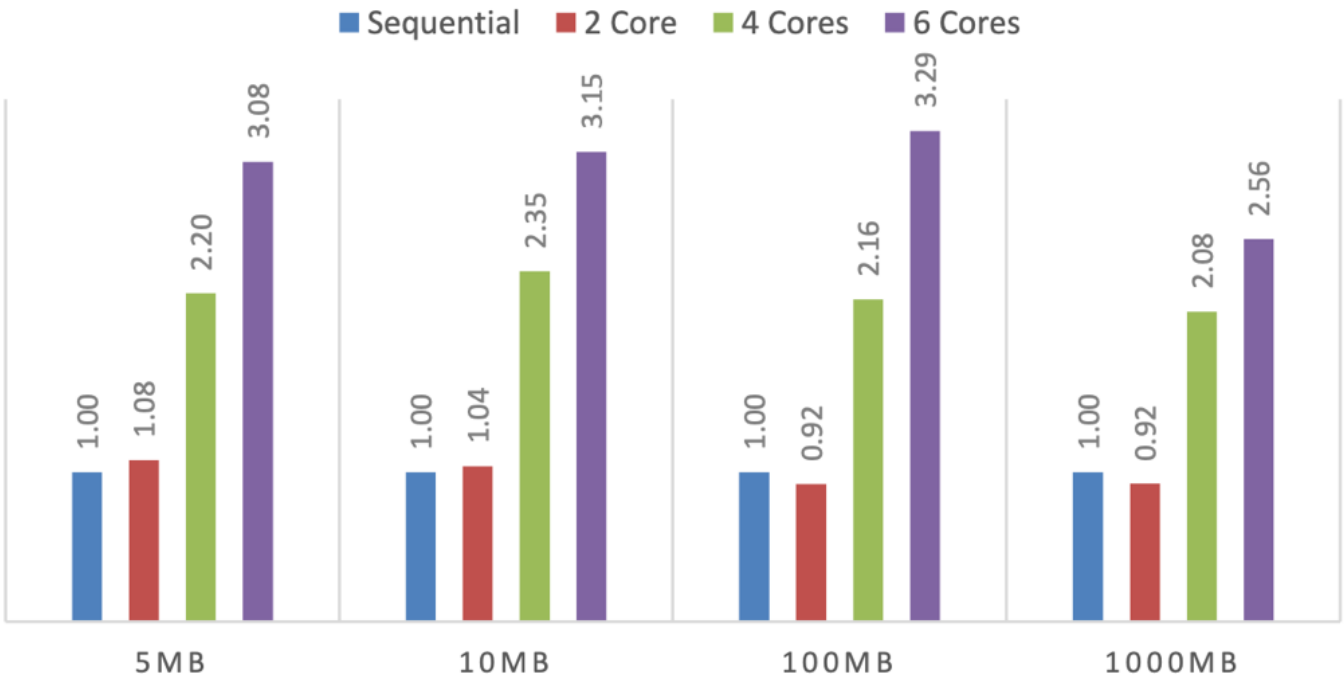
AES Speedup	5MB	10MB	100MB	1000MB
Sequential	1.00	1.00	1.00	1.00
2 Core	1.08	1.04	0.92	0.92
4 Cores	2.20	2.35	2.16	2.08
6 Cores	3.08	3.15	3.29	2.56

AES Efficiency	5MB	10MB	100MB	1000MB
Sequential	1.00	1.00	1.00	1.00
2 Core	0.54	0.52	0.46	0.46
4 Cores	0.55	0.59	0.54	0.52
6 Cores	0.51	0.52	0.55	0.43

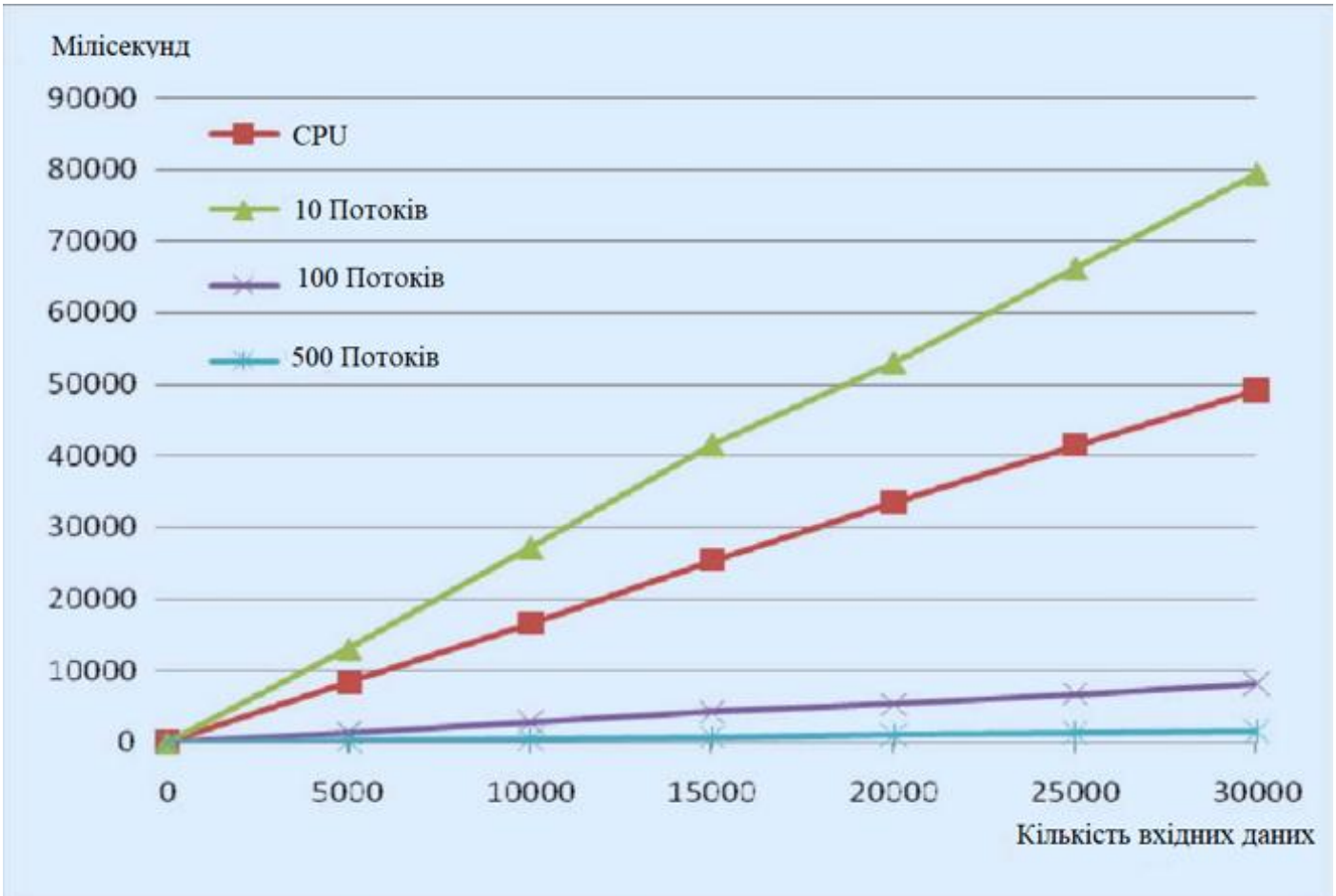
ПЛАКАТ 4 СХЕМА ЧАСУ ВИКОНАННЯ ПОСЛІДОВНОЇ ТА ПАРАЛЕЛЬНОЇ РЕАЛІЗАЦІЇ РІЗНИМИ ПРОЦЕСОРНИМИ ЯДРАМИ В МІЛІСЕКУНДАХ ДЛЯ AES АЛГОРИТМУ



ПЛАКАТ 5 СХЕМА ПРИСКОРЕННЯ АЛГОРИТМУ AES ПРИ РЕАЛІЗАЦІЇ РІЗНИМИ ПРОЦЕСОРНИМИ ЯДРАМИ



ПЛАКАТ 6 ПОРІВНЯННЯ ЧАСУ ВИКОНАННЯ АЛГОРИТМУ RSA В РІЗНИХ РЕЖИМАХ



ПЛАКАТ 7 ПОРІВНЯННЯ ПОСЛІДОВНОГО ТА ПАРАЛЕЛЬНОГО ВИКОНАННЯ АЛГОРИТМУ RSA